Adoga, Haruna Umar (2024) *Leveraging NFV heterogeneity at the network edge.* PhD thesis.

https://theses.gla.ac.uk/84344/

Copyright and moral rights for this work are retained by the author

A copy can be downloaded for personal non-commercial research or study, without prior permission or charge

This work cannot be reproduced or quoted extensively from without first obtaining permission from the author

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given

Enlighten: Theses
https://theses.gla.ac.uk/
research-enlighten@glasgow.ac.uk

Leveraging NFV Heterogeneity at the Network Edge

Haruna Umar Adoga

Submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy

School of Computing Science

College of Science and Engineering
University of Glasgow

January 2024

© Haruna Umar Adoga

Abstract

With network function virtualisation (NFV) and network programmability, network functions (NFs) such as firewalls, traffic load balancers, content filters, and intrusion detection systems (IDS) are virtualized and either instantiated on user space hosts using virtual machines (VMs), lightweight containers, or in the network data plane using programmable switching technology such as P4 or offloaded onto Smart network interface cards (NICs) – often chained together to create a service function chain (SFC), based on defined service level agreement (SLA). The need to leverage heterogeneous programmable platforms to support the in-network acceleration of functions keeps growing as emerging use cases come with peculiar requirements. This thesis identifies various heterogeneous frameworks for deploying virtual network functions that network operators can leverage in service provider networks. A novel taxonomy that provides network operators and the wider research community valuable insights is proposed. The thesis presents the performance gains obtained from using heterogeneous frameworks for deploying virtual network functions using real testbeds. In addition, this thesis investigates the optimal placement of vNFs over the distributed edge network while considering the heterogeneity of packet processing elements. In particular, the work questions the status quo of how vNFs are currently being deployed, i.e., the lack of frameworks to support the seamless deployment of vNFs that are implemented on diverse packet processing platforms – leveraging the capability of the programmable network data plane. In response, the thesis presents a novel integer linear programming (ILP) model for the hybrid placement of diverse network functions that leverages the heterogeneity of the network data plane and the abundant processing capability of user space hosts, with the objective function of minimizing end-to-end latency for vNF placement. A novel hybrid placement heuristic algorithm, HYPHA, is also proposed to find a quick, efficient solution to the hybrid vNF placement problem. Using optimal stopping theory (OST) principles, an optimal placement scheduling model is presented to handle dynamic edge placement scenarios. The results in this work demonstrate that employing a hybrid deployment scheme that leverages the processing capability of the network data plane yields minimal user-to-vNF latency and overall end-to-end latency while fulfilling the placement of a diverse set of user requests from emerging use cases to speed up service delivery by network operators. The results also show that network operators can leverage the high-speed, low-latency feature of data plane packet processing elements for hosting delay-sensitive applications and improving service delivery for subscribed users. It is shown that the proposed hybrid heuristic algorithm can obtain near-optimal vNF mapping while incurring fewer latency threshold violations set by network operators. Furthermore, in addition to emerging edge use cases, the placement solution presented in this thesis can be adapted to place network functions efficiently in core network infrastructure while leveraging the heterogeneity of servers. The dynamic placement scheduler also minimises the number of latency violations and vNF mi-

grations between heterogeneous hosts based on SLAs set by network operators.

Acknowledgements

In the Name of God the Merciful, the Compassionate, بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

I want to thank my supervisors, Professor Dimitrios P. Pezaros and Dr Yehia Elkhatib. I immensely appreciate Professor Pezaros for always providing critical feedback and guidance throughout my PhD and his genuine concern for my general well-being. Thank you to the Petroleum Technology Development Fund (PTDF), Nigeria, for funding my studies. To my loving wife (Fatsuma Abubakar - *my beautiful Tulip*) for all your patience, love and support throughout the years and to my *li'l man* Arif Haruna Adoga, who came to this world during this PhD journey – I love you both.

This work is dedicated to my parents, Hajiya Rakiya Umaru Adoga and the late Alhaji Umaru Adoga, who is no longer with us to witness this remarkable feat by his son. May Allah shine light and infinite mercy in my father's grave and forgive all his shortcomings, *Ameen*. May Allah bless my mother and forgive all her shortcomings; thank you for all your love, support and prayers and for regularly checking on your *dan auta*; I love you. To my brothers Suleiman, Mohammed, Abdullahi, Usman, and Yusuf Umar Adoga and my sisters Maryam, Khadija, and Fatima Umar Adoga, late Aisha and Habiba Umar Adoga, thank you for always being there for me; I love you all. To Suleiman Umar Adoga, thank you for all you have done and continue to do for me! I wouldn't have made it this far without you; I love you!

I want to thank Dr Christos Anagnostopoulos and Dr Lewis McKenzie (my Annual Progress Review panel reviewers) for always providing helpful feedback on my work over the years. Thank you to all the anonymous reviewers who have helped shape my research in one way or another. To Dr Mircea Iordache-Sica, Dr Kyle Simpson, and Dr Stefanos Sagriotis, thank you for helping me keep track and maintain focus, especially during the pandemic's peak with the daily standup and occasional gaming and drinks. Thanks for always answering my queries and providing regular feedback on my work. Thanks, Engr Alhamdu Umaru Bello, for all your selfless support and friendship throughout the years.

To Dr Ibrahim Alghamdi (Abu-Aws), thanks for showing me the ropes when I first arrived in Glasgow and for your support throughout my PhD journey. Thank you, Dr Yousef Alhaizaey, for all your help and support through the years. Thanks to Dr Lucky Onwuzurike, Zayyad Abdulhameed Girie, Dr Muhammad Dangana, Dr Halilu Sale, and Abdullahi Abubakar Dauda for all your support and friendship. To all the amazing people I shared an office with at any given time at the Sir Alwyn Williams Building: Dr Stephen McQuistin, Dr Vivian Band, Dr Mihail Yanev, Thomas Janich, Dr Hatem Aied S Almazarqi and Xicheng Li. Thank you for all the support. To all the wonderful folks at the admin and teaching offices of the School of Computing Science I have worked with over the years as a Tutor, thank you for all your support.

“The heart’s obtainment of knowledge is like the body’s nourishment from food. So just as the bodies feel the effects of food and drink, the hearts feel the effects of that which they are fed from the types of knowledge and sciences, which is their food and drink.”

Ibn Qayyim Al-Jawziyya (1292 CE – 1350 CE)

Contents

1 Introduction **1**

1.1 Overview 1

1.2 Thesis Statement 4

1.3 Contributions 4

1.4 Thesis structure and organization 5

2 Literature Review **7**

2.1 Chapter Overview 7

2.2 SDN, NFV, and SFC 7

2.2.1 Software-Defined Networking 8

2.2.2 SDN and data plane programmability 9

2.2.3 Why data plane programmability? 11

2.2.4 Network Function virtualisation 14

2.3 The Chaining of Virtual Network Functions 18

2.4 NFV Deployments and Performance – A taxonomy of frameworks 20

2.5 Leveraging the Edge Computing Paradigm for VNF Placement 25

2.6 Summary 27

3 On the Performance Benefits of Heterogeneous Virtual Network Function Execution Frameworks **29**

3.1 Overview 29

3.1.1 The Need for Heterogeneous Packet Processing Elements 30

3.2 Heterogeneous VNF Frameworks 31

3.2.1 VNFs on Containers, Commodity Servers and Kernel-Bypass . . . 31

3.3 VNF Frameworks Performance Evaluation 38

3.3.1 Data plane Packet Processing Considerations 39
3.3.2 Virtual Network Functions Testbed Implementations 40
3.3.2.1 Experimental setup 40
3.3.2.2 Traffic generation and monitoring 42
3.3.2.3 Network functions instantiation 43
3.4 Heterogeneous Frameworks Evaluation 45
3.4.1 DPDK and containerised functions performance 45
3.4.2 Software switch data plane functions performance 45
3.4.3 KVM-based functions performance 46
3.4.4 VNF Suitability and Recommendations 48
3.5 Summary 50

4 Latency-aware VNF Placement on Heterogeneous Edge Hosts 51
4.1 Overview 51
4.2 Design Considerations 52
4.2.1 The Need for Hybrid VNF Placements 52
4.2.2 System Operational Overview 57
4.2.3 High-Level Design Requirements 58
4.2.4 Provision for Resilience, Elasticity, and Scalability 59
4.2.5 VNF Performance Considerations 60
4.2.6 Considerations for Large-scale Deployments 61
4.3 Latency-aware Hybrid VNF Placement 62
4.3.1 Rationale 62
4.3.2 Hybrid Placement System Model 63
4.3.3 Placement Problem Formulation 65
4.4 Hybrid Placement Heuristic Algorithm (HYPHA) 66
4.4.1 Heuristic Computational Complexity 69
4.5 Dynamic Hybrid Placement Scheduling 70
4.5.1 Rationale for dynamic hybrid placement scheduling 70
4.5.2 Dynamic Hybrid Placement 72
4.5.3 Dynamic Optimal Placement Problem 72

4.5.4 Optimal Stopping Theory Concepts 73
4.5.5 Definitions and theorems 74
4.5.6 Hybrid VNF Placement – Optimal Scheduling 75
4.6 Summary 76

5 Edge Hybrid VNF Placement Framework Evaluation **78**
5.1 Overview 78
5.2 Evaluation Environment 79
5.2.1 User-space-only and hybrid placement latency 81
5.2.2 Optimisation and heuristic objective functions 82
5.2.3 Number of placement requests fulfilled 84
5.2.4 Dynamic latency violations 86
5.2.5 Hybrid Placement Scheduling With Optimal Stopping Time 87
5.2.6 Number of VNF Migrations 88
5.3 Summary 89

6 Conclusion and Future Work **91**
6.1 Overview 91
6.2 Contributions 91
6.3 Thesis Statement Revisited 93
6.4 Further Use Cases 94
6.5 Future Research 96
6.5.1 In-band Network Telemetry and Multi-domain Service Orchestration 97
6.5.2 Intelligent Hybrid Function Chaining at the Network Edge 98
6.6 Concluding Remarks 99
6.7 Publications 101

Bibliography **102**

List of Tables

2.1 Commonly used virtual network functions [3] 17

3.1 Virtual Network Functions Operations and Classification [10] 31

3.2 Mean bandwidth rates (in Mbps) for data-plane and VNF deployments. . . 47

4.1 System model parameters 64

5.1 Common use case latency characteristics 80

List of Figures

2.1 Packet flow in the OpenFlow pipeline. 9

2.2 The OpenFlow and programmable control/data plane separation. 11

2.3 The P4 packet processing pipeline 13

2.4 The ETSI NFV Architecture 15

2.5 NFV/SFC frameworks taxonomy [3]. 21

3.1 Virtual network functions on VMs, Containers and Kernel-Bypass frameworks 32

3.2 Docker and kernel libraries 33

3.3 Network Functions using kernel-bypass with DPDK 35

3.4 An overview of the operation of I/O-bound NFs, e.g., ACL/Firewall, Header Classifier, and caching. Tasks involve IP header lookup and data store access. 37

3.5 An overview of the operation of compute-bound NFs, e.g., NAT, IDS and QoS optimization. Tasks carried out involve more complex types of processing performed on packets. 38

3.6 VNF frameworks performance evaluation testbed – packets are generated and made to traverse one or more diverse frameworks simultaneously. Match-action rules are written based on the type of functionality (e.g., a router or a firewall vNF). 40

3.7 Million Packets Per Second – DPDK and Containerized vNFs. All NFs are tied to separate CPU cores for performance isolation and measurement consistency. 46

3.8 Mean bandwidth rates for switch data plane and KVM-based deployments of network functions. 47

3.9 CPU utilization – switch data plane firewall and router maintain much lower CPU utilization when compared with VM functions, as the million packets per second increases. 48

4.1 Edge Network Infrastructure 57

5.1 Average user to vNF latency in user-space-only and hybrid optimisation placement scenarios. 81

5.2 Heuristic and optimisation objective functions for different placement scenarios. 83

5.3 Probability distribution functions of stateless and stateful vNFs 84

5.4 Comparison of the number of fulfilled vNF placement requests in hybrid and data plane-only scenarios, with different numbers of user placement requests. 85

5.5 Heuristic and optimisation latency violations performance. 86

5.6 New placement scheduling is dynamically triggered before getting to the latency violation threshold. 88

5.7 Total number of vNF migrations using Optimal Stopping Time and scheduling at every 50-time instances. 89

Abbreviations:

Abbreviation	**Meaning**
API	Application programming interface
CAPEX	Capital expenditure
CL	Classifier
CNF	Cloud-native function
DMA	Direct memory access
DPI	Deep packet inspection
eNF	embedded network function
ETSI	European Telecommunication Standards Institute
FPGA	Field-programmable gate array
EPC	Evolved packet core
ITU	International telecommunication union
IDS	Intrusion detection system
ILP	Integer linear programming
IoT	Internet of Things
IPS	Intrusion prevention system
P4	Programming Protocol-independent Packet Processors
PCAP	Packet capture
PoC	Proof of concept
PDP	Programmable Data Plane
RSS	Receive side scaling
SC	Service Chain
SDN	Software-defined networking
SFF	Service function forwarder
SP	Service provider

SF	Service function
SFC	Service function chaining
SFCC	Service function chaining controller
MAC	Media access control
ML	Machine learning
NAT	Network address translation
NFV	Network function virtualization
NSH	Network service header
NFVI	NFV infrastructure
NFVO	NFV Orchestrator
NS	Network service
NFF	Network function forwarder
OF	OpenFlow
OPEX	Operational expenditure
QoS	Quality of service
QoE	Quality of experience
vCPE	virtual customer premises equipment
VNE	Virtual network embedding
VoIP	Voice over Internet Protocol
vNF	Virtual network function
VNFC	Virtual network function component
VIM	Virtual infrastructure manager

Chapter 1

Introduction

1.1 Overview

As the number of devices needing Internet connectivity continues to grow, with large-scale adoption of IoT technologies and innovative use cases, the underlying network infrastructure's efficiency becomes critical. According to a recent estimate [1], at least 24 billion interconnected IoT devices are expected by the year 2050, which range from refrigerators, thermostats, healthcare wearables, Industrial Control Systems, elevators, etc. This exponential growth in interconnected devices is also fueled by the increased computational, storage and memory capacities, reduced cost of smartphones and single-board computers (SBCs) in edge network environments.

To effectively provision the underlying network infrastructure that supports the growth in the number of interconnected devices, in the early years of the 21st century, both the research and industry communities have moved away from the use of fixed-function hardware by adopting a new technology that essentially separates the control plane (i.e. the packet forwarding and routing logic component) from the hardware data plane (responsible for packet forwarding) of networking devices. This brought about software-defined networking (SDN) – a significant paradigm shift in networking.

To further support the logical separation of the control and data plane brought about by SDN, the fixed-function networking hardware needed to be virtualised; thus, network function virtualisation (NFV) was introduced as a way of creating virtual network functions (VNFs) [2,3] that can easily be implemented and orchestrated using an SDN controller (see §2.2.1). As we will see in much greater detail later in this thesis (§2.2), NFV deals with the virtualisation of hardware middleboxes, i.e., a departure from using traditional fixed-function networking hardware, which, by implication, are proprietary, expensive, and difficult to extend or implement new functionalities [3].

With the advent of software-defined networking (SDN) and network function virtualisation (NFV), middlebox functionality is increasingly being virtualised and provided in software, which can reduce power consumption, resource usage, and operational costs for service providers [4, 5]. This paradigm is a departure from using hardware middleboxes, often proprietary and thus not easily extendable by service providers. By abstracting network functionalities and implementing them in software, network operators can create network functions that suit their service level agreement (SLA) and service models [6].

Service function chaining (SFC) (see §2.3) is generally considered one of the important use cases of NFV, SDN and network programmability [7], which is also made possible using a centralised network controller that has a *global* logical view of the entire network infrastructure, and handles tasks such as the creation of service chains and orchestration of traffic between vNFs [8]. Regarding the location of network functions in a service chain, the virtualised infrastructure can span multiple data centres, which calls for inter-data-centre networking or within the same data centre, which results in an intra-data-centre network.

There are diverse frameworks for implementing vNFs in service provider network environments; these frameworks can either implement functions in the data plane, using programmable switching technologies such as P4 [9] or offloaded onto SmartNICs, or using user space hosts [10]. Although the network data plane offers high-speed packet processing, it is limited in key ways. It has a limited memory footprint, lacks *in-situ* packet header manipulation support, has poor state management support, and does not readily support manipulating floating-point values [11] etc. The network data plane is generally resource-constrained and thus does not accommodate vNFs with high storage or scalability requirements due to the low memory footprint [12].

The network's data plane serves as a suitable alternative for implementing VNFs. Hence, network operators can leverage the data plane and process-based NFV frameworks to deploy softwarized network functions. Doing so has the benefits of reducing the cost of procuring high-end servers for NFV deployment and the overheads for compute-bound network functions. Typically, VNF deployment is carried out using VMs [13], containers [14], Unikernels [15], Click-based processing elements, and data plane Development Kit (DPDK), which uses zero-copy and kernel bypass to speed up packet processing [16].

Due to the diverse nature of emerging edge computing applications, a user space-only or data plane-only deployment does not efficiently cater to emerging use cases such as e-healthcare, self-driving cars, VR/MR/AR, etc, which is one of the key limitations of the existing VNF placement solutions (presented in Chapter 2). A hybrid solution can leverage the advances from the data plane and user space. The need to leverage heterogeneous programmable platforms to support the in-network acceleration of functions keeps growing as emerging use cases come with particular latency requirements.

Another limitation of the current literature is the lack of frameworks for achieving end-to-end programmability across the packet processing pipeline at the network edge. To achieve this, we must design placement frameworks to leverage the heterogeneity of data plane devices along a given network path. Thus, frameworks that allow us to choose the right packet processing elements are desirable based on their suitability to host a particular vNF.

As a deviation from how VNFs are currently being placed in most deployments, i.e., the lack of frameworks to support seamless implementation of functions on diverse platforms, a placement scheme that provisions network functions in the programmable data plane and user space hosts to reduce packet processing overhead and overall path latency. Virtual network functions are either being placed entirely on user space processing elements such as [17–19] or using only the network data plane as demonstrated in [20–24]. Some VNF deployments are done using packet processing acceleration frameworks as demonstrated in OpenNetVM [16], FlexiIPS [25], Phantom SFC [26] and [27]. This thesis proposes a framework that allows network operators to place VNFs on both data plane, user space, and packet processing acceleration platforms to reduce packet processing overhead, latency, and resource utilization, which also helps service providers meet the agreed SLA with their subscribers.

This thesis thoroughly reviews the state-of-the-art NFV and SFC deployment frameworks in service provider environments and proposes a novel taxonomy of frameworks [3]. This informed the decision on the direction of the remainder of the thesis, i.e., to further explore options for deploying virtual network functions that help network operators improve overall service delivery by efficiently meeting customers' SLAs. The thesis argues for the need to leverage the capability of the network data plane at the edge for improved service delivery. It encourages why data plane programmability should not be left for the data centre (an argument for end-to-end programmability at the edge). The testbeds designed are ideal for comparative performance results while using different packet processing elements in hosting network functions [10].

The thesis investigates the optimal placement of virtual network functions at the distributed network edge. It presents a detailed analysis of the need for hybrid vNF placement on heterogeneous packet processing elements, including some relevant emerging use cases. A novel ILP model for the hybrid placement of diverse network functions on the heterogeneous data plane and user space hosts is also presented, with the objective function of minimizing end-to-end latency and leveraging the network data plane for vNF placement. A novel hybrid placement heuristic algorithm, HYPHA, is also presented to efficiently find a quick solution to the hybrid vNF placement problem and presents an in-depth evaluation of the proposed placement model and heuristic algorithm using actual service provider network conditions.

The performance of different VNF implementation alternatives was evaluated. The thesis

motivates the deployment of network functions as part of the programmable data plane of the network for better service delivery, primarily where latency-sensitive and bandwidth-intensive network functions are to be implemented. Through extensive evaluations using simulated real-world network topology, the thesis demonstrates that employing a hybrid deployment scheme that leverages the processing capability of the network data plane yields minimal overall end-to-end latency and fulfils the placement of a diverse set of vNF requests. Network operators can leverage the high-speed, low-latency feature of data plane packet processing elements for hosting delay-sensitive applications and improving service delivery for subscribed users.

The model presented will help network operators efficiently place virtual network functions in a hybrid manner while reducing end-to-end latency. This will result in faster packet processing based on the number of network functions that can be pushed down to the data plane. The hybrid placement algorithm proposed in this work can handle a more extensive network topology and return placement results within the shortest possible time, which makes it ideal for the nature of edge network environments.

1.2 Thesis Statement

This thesis explores the different frameworks for implementing virtual network functions, including the programmable data plane, user space, and packet acceleration frameworks. **The thesis advocates advancing function virtualisation and deployment by exploiting diverse emerging environments, such as programmable switch hardware and in-network acceleration platforms, for VNF placement at the network edge.** Using extensive evaluations, key metrics such as end-to-end path latency, bandwidth, and resource utilization in the edge network environment demonstrated the promising benefits of leveraging heterogeneous packet processing elements in vNF deployments in next-generation edge networks to improve the overall QoS and SLA for subscribed users.

1.3 Contributions

The contributions of this thesis are:

Chapter 2:

- A comprehensive state-of-the-art review of the relevant literature in vNF and SFC deployments to build a strong understanding of the most recent efforts in this domain. A novel taxonomy of these frameworks is presented.

Chapter 3:

- A detailed study into the nature of virtual network functions and the heterogeneity of packet processing elements provides insights into the diverse capabilities of such processing elements.
- An in-depth analysis of the composition and complexity of SFCs and edge placement environments, including a classification of representative virtual network functions used by network operators in next-generation networks (§3.1.1). The design of real testbeds ideal for evaluating the performance of heterogeneous virtual network functions using diverse packet processing elements is presented in §3.3.
- An evaluation of the performance benefits of leveraging heterogeneous packet processing elements for implementing virtual network functions in next-generation networks (§3.4).

Chapter 4:

- A formulation for latency-aware hybrid placement of virtual network functions and an exact solution using Integer Linear Programming and the Gurobi optimisation solver (§4.3.2).
- A proposal of an optimal placement scheduling model using optimal stopping theory (OST) principles to cater to dynamic hybrid edge placement scenarios, minimising cumulative latency violations and cost of vNF migration between heterogeneous hosts (§4.5).
- The hybrid placement heuristic algorithm designed for placing virtual network functions on diverse packet processing elements at the network edge to better utilize available resources and offer faster service delivery by network operators (§4.4).

Chapter 5:

- An evaluation of the performance of the proposed dynamic hybrid placement model and heuristic algorithm, using real-world latency characteristics, over a simulated nationwide network topology.

1.4 Thesis structure and organization

The remainder of this thesis is structured as follows:

- **Chapter 2** presents the outcome of the in-depth review carried out in SFC and NFV provisioning in service provider environments, using various frameworks and the current state-of-the-art in this domain. It also presents the important requirements for the composition of virtual network functions to create SFCs and the challenges in achieving resilient and scalable deployments. A shift from *vanilla* SDN to using programmable data planes is also discussed in the context of SFC provisioning. It also shows the challenges that come with the status quo of deploying virtual network functions and service function chains in general.

- **Chapter 3** presents a detailed analysis of some of the problems identified in Chapter 2 with the design of a real testbed for the deployment and performance evaluation of heterogeneous virtual network functions frameworks. The chapter also presents evaluation results and key recommendations for network operators based on the suitability of the packet processing elements used in the experiment.

- **Chapter 4** presents the design of a novel dynamic hybrid vNF placement optimisation model and heuristic that leverages the low-latency, high-speed feature of the network data plane and the abundant resources on user space hosts. The prototype implementation environment for the hybrid placement model and the heuristic algorithm are also presented. The chapter concludes with implementation details of some real-world example functions and the details of the real-world network topology used.

- **Chapter 5** presents a comprehensive evaluation of the designs in Chapter 4. The details of the evaluation environment are presented, including the performance gains of using heterogeneous vNF frameworks. This chapter presents the performance of the proposed dynamic hybrid placement model and heuristic algorithm, using real-world latency characteristics, over a simulated nationwide network topology.

- **Chapter 6** presents a summary of the findings and contributions of this thesis. The initial thesis statement is also revisited in this chapter. Finally, future research directions are also presented in this chapter.

Chapter 2

Literature Review

2.1 Chapter Overview

This chapter presents an in-depth review of the state-of-the-art in next-generation networks, which involves developments such as network softwarisation, data plane programmability, packet processing acceleration and offloading technologies, and edge computing. The chapter looks at the contributions of the research community in terms of the gradual shift from traditional SDN to data plane programmability, the technologies involved, and the frameworks used for deploying vNFs at the distributed edge network. It presents the essential requirements for the composition of vNFs to create SFCs and the challenges in achieving resilient and scalable vNF deployments. It also shows the challenges that come with the status quo of deploying virtual network functions and service function chains in general, which neatly leads the reader to the contributions of this thesis and the work presented in subsequent chapters.

2.2 SDN, NFV, and SFC

Computer networks have grown tremendously, with more devices requiring Internet access. Emerging use cases such as the Internet of Things (IoT), self-driving cars, drones and growing consumer electronics (e.g., smart home appliances) are just a few examples of devices that need somewhat constant Internet connectivity. This high demand has led to several innovations by network engineers and service providers to provide scalable and resilient network infrastructure that can support diverse use cases [28].

Network slicing is a key component of next-generation networks, mainly because an inherent part of 5G/6G standardisation is the ability for network operators to migrate some or all of their network services to virtual network infrastructure, thereby reducing both capital and

operational costs. With advances in network function virtualisation (NFV) and software-defined networking (SDN), network functions (NFs) such as firewalls, traffic load balancers, content filters, and intrusion detection systems (IDS) etc., are either instantiated on packet processing elements such as virtual machines (VMs), lightweight containers, SmartNICs, programmable switches etc. often chained together to create a service function chain (SFC) that meets subscribers and operator-defined service level agreements (SLAs).

2.2.1 Software-Defined Networking

SDN decouples the control plane from the data plane in the networking devices such as routers and switches. Traditional non-SDN networks often have control and data planes integrated into a single device, which brings challenges such as management complexity and scalability issues. Implementing centralised network control using SDN controllers results in easier service deployment and management [29]. This helps network operators easily steer traffic between NFs by scaling across multiple physical machines.

The functional separation of the network infrastructure into control and data planes is the core concept behind SDN [30]. The application layer consists of various network applications that provide network services that use the Northbound Interfaces to send requests to the control plane (centralised logical control). A global view of the network infrastructure is maintained by an SDN controller such as OpenDaylight [31], POX [32], RYU [33], or a custom-built controller that can be used to manage network functions, which handles requests from the network applications and sends instructions to the network's data plane for packet processing [34].

Using the southbound application programming interfaces (APIs) and the OpenFlow protocol, rules are sent down to devices in the network's data plane responsible for packet processing and forwarding [30].

SDN-aware virtual switches such as the Open vSwitch (OvS) have a data-plane component that handles traffic forwarding. Regarding NFV implementations, the network data plane can be represented using a production-grade virtual switch [30]. Virtual switches can be configured to emulate typical middlebox functionalities such as firewall/access control, Network Address Translation (NAT), routing, and switching. When functioning as a router, it matches the longest source and destination IP prefix and forwards packets using specified ports.

As a common example, a typical firewall functionality matches IP addresses and specific TCP/UDP port numbers and denies or permits packets (based on match-action flow rules). As depicted in Figure 2.1, actions are taken on ingress packets based on the rules defined in the table entries; the Group Table allows for the definition of groups that encapsulate

sets of actions, enabling more advanced packet handling and forwarding decisions when needed [35], before finally going through the egress processing stage.

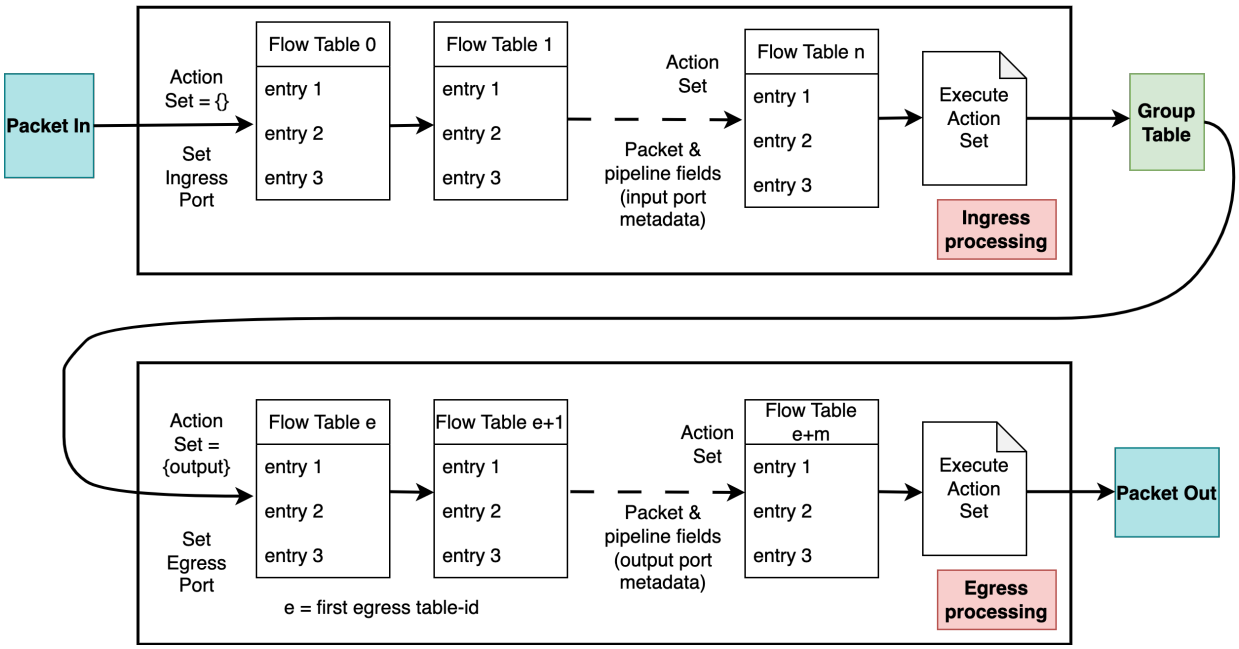

Figure 2.1: Packet flow in the OpenFlow pipeline.

Although traditional SDN networks use the OpenFlow protocol to communicate with the network data plane by inserting flow rules on devices, the network has become more programmable. Programmability allows network operators to define the processing pipeline and how packets are processed using high-level languages such as P4 [9]. Figure 2.1 depicts the OpenFlow packet processing pipeline.

2.2.2 SDN and data plane programmability

In service provider networks, creating, deleting, modifying, and steering traffic in SFCs is carried out efficiently using SDN and NFV technologies [36]. These technologies are the key networking paradigms at the core of the frameworks surveyed in our study. In this section, these technologies are described as they relate to NFV/SFC implementation frameworks in service provider network environments, thus showing their interrelation in the operations of next-generation networks. Even though chaining hardware middle-boxes is possible, using NFV makes it much easier and cheaper [29]. Thus, SDN is employed for orchestrating virtual network functions by providing centralised logical control and the creation of service chains.

SDN and data plane programmability are closely related, as separating the control and data planes in SDN enables greater data plane programmability [37]. It is essential to note that,

as one of the key distinctions, SDN allows the network operator to control how the network is configured (e.g., communication between switches and controllers using the OpenFlow protocol) while at the same time, data plane programmability controls how the packets are handled, traversing the processing pipeline.

In an SDN architecture, the centralised controller can be programmed to automate network device configuration and respond to network conditions changes. This allows network operators to easily manage and automate the configuration of their networks rather than having to configure each device manually [28, 38].

Unlike the traditional SDN setup that uses the OpenFlow protocol which has built-in actions and some protocols [30], programmable switches are more flexible and truly arbitrary because they can be programmed to perform a broader range of tasks. For example, a programmable switch can be programmed to implement new routing protocols, traffic-shaping algorithms, or security policies, such as advanced DDoS prevention/mitigation [39]. On the other hand, OpenFlow switches are limited to the set of tasks supported by the OpenFlow protocol, e.g., switch-to-controller communication, etc.

Figure 2.2 depicts a high-level comparison of the operation of OpenFlow SDN and programmable data planes, where P4 programs are written and compiled and then deployed on programmable switches. This allows for creating a highly programmable pipeline instead of a fixed (less flexible) processing pipeline found in OpenFlow SDN.

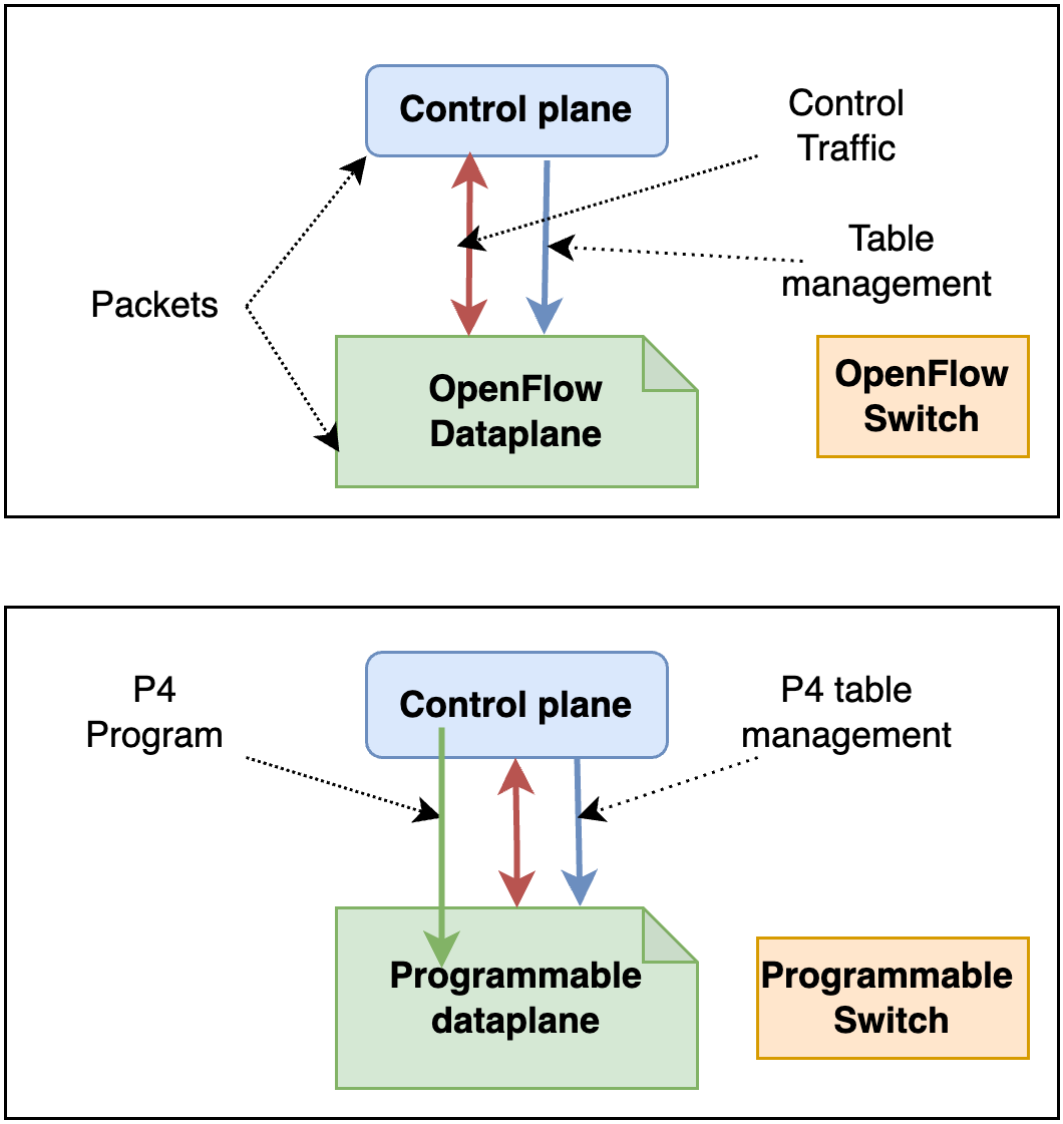

Figure 2.2: The OpenFlow and programmable control/data plane separation.

2.2.3 Why data plane programmability?

Data plane programmability aims to allow operators to flexibly define the packet processing pipeline [40]. This paradigm shift has the potential to improve network flexibility, scalability, and manageability significantly and has been closely tied to the emergence of Software-Defined Networking (SDN). Traditionally, network devices such as routers and switches have been configured manually to deliver specific customer services, which essentially use fixed-function architectures. Data plane programmability addresses this problem by allowing operators to quickly write target-specific, custom-built network functions using readily available APIs and languages such as P4 [38].

One widely used high-level language for achieving data plane programmability is the P4 language, briefly discussed in 2.2.2. As depicted in Figure 2.3, the P4 packet processing pipeline

is a software-defined architecture that allows network engineers to program the behaviour of the data plane of network devices efficiently. The pipeline consists of several stages, each performing a specific task on the packets traversing the pipeline.

Looking at Figure 2.3, the process starts when network operators write a high-level P4 code that defines a specific function (e.g., a router); this is then compiled into a target-specific binary and then deployed onto the programmable data plane. The parser decodes the packet header and extracts the relevant information. Note that the parser is typically implemented as a finite state machine [9], which allows it to handle various packet formats.

Once the packet header has been decoded, the next stage in the pipeline is the match-action table. The match-action table is a data structure that contains a set of rules that define how to process the packet. Each rule in the table consists of a set of match conditions and actions. The match conditions determine which rule should be applied to the packet. The actions are the operations that will be performed on the packet if a particular rule is matched [9].

If the packet matches a rule in the match-action table, the actions associated with that rule will be executed. The actions can modify the packet header, forward the packet to a different destination, or drop the packet – depending on defined rules. The controller can remove and add table entries or perform extern control [38]. The final stage in the pipeline is the deparser. The deparser is responsible for reassembling the packet header and trailer and sending the packet out of the egress network interface.

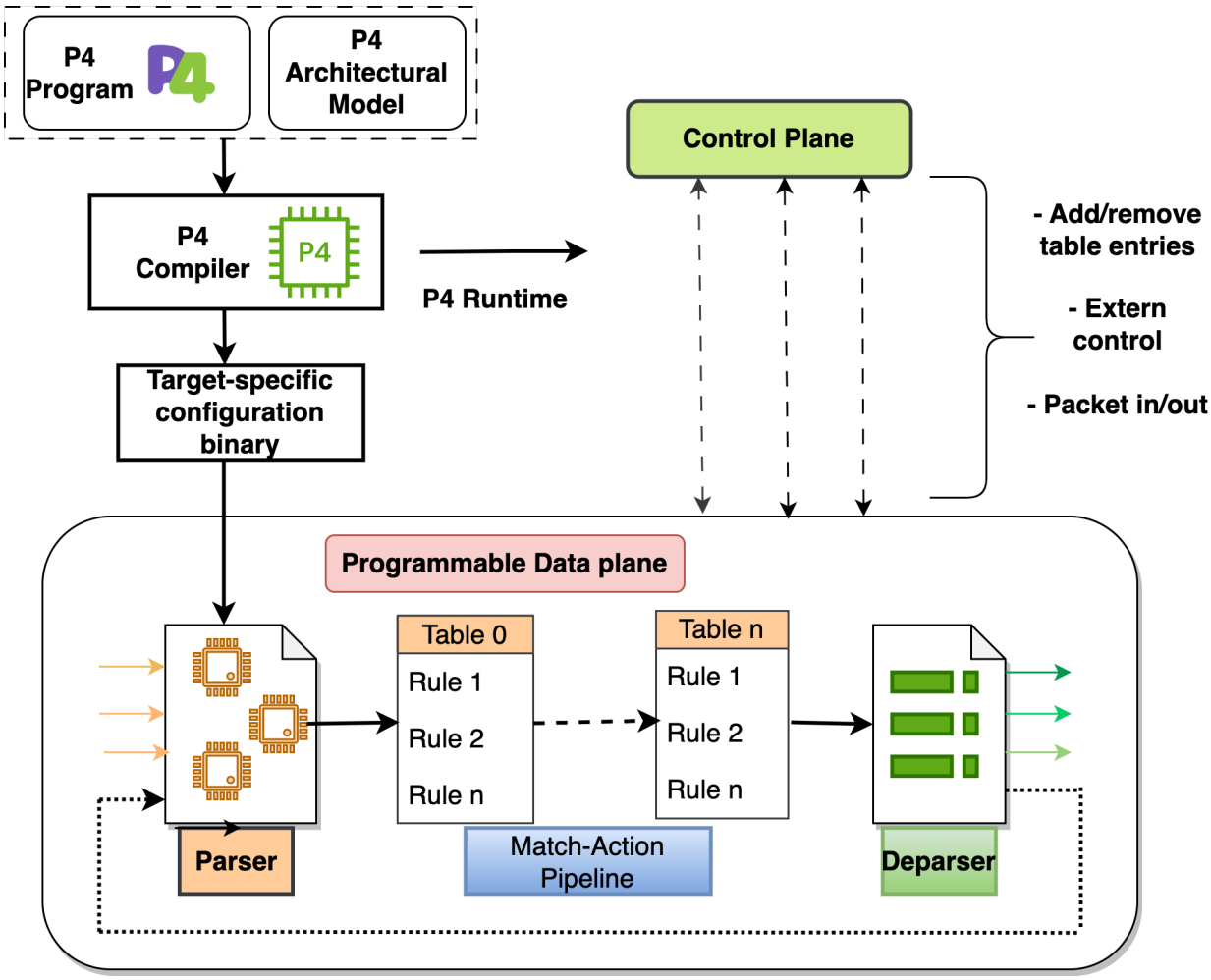

Figure 2.3: The P4 packet processing pipeline

Here are some benefits of using the P4 packet processing pipeline for achieving data plane programmability. [41]:

- Scalability: The P4 packet processing pipeline can be scaled to support large networks with high traffic volumes. By providing network operators with a flexible, efficient, and scalable way to program target-specific network devices, the P4 packet processing pipeline can help improve network performance, security, and reliability [2]. Using PDP hardware, for example, some P4-enabled use cases, such as In-network computing, can be made scalable. This concept gives network operators the flexibility to easily offload parts of packet processing operations to programmable hardware devices.

- Flexibility: With data plane programmability, network operators can easily define network functionalities such as NAT, DDoS, ACL, etc., and compile them into target-specific binary, which makes the entire process of building tailored network functions flexible. The P4 language, for example, allows network operators to express complex

network functions concisely and efficiently [42]. Being a protocol-independent language also provides network operators with the flexibility of quickly building custom APIs and functions that are target-agnostic [2].

- Portability: The P4 language is designed to be portable, meaning that P4 programs can be compiled and run on various vendor-neutral hardware platforms. Using the Portable NIC Architecture (work in progress), the support for table modification at runtime aids the portability of P4 programs [38]. This makes deploying P4 programs on many network devices possible, from small home routers to large enterprise switches and FPGAs.

- Modularity: Data plane programmability allows for the modular design and deployment of network functions. The P4 packet processing pipeline is designed to be modular, meaning that P4 programs can be easily divided into smaller, self-contained modules. This makes it easier to develop and maintain P4 programs and allows it to reuse modules in different network applications [38]. The modularity support offered by P4 also makes it possible to support multiple software and hardware programmable targets; some common examples are NetFPGA cards, software switches like PISCES [43] and bmv2 [44], SmartNICs, etc.

Overall, using a programmable packet processing pipeline is a powerful and versatile approach that can be used to implement diverse network functions. By providing network operators with a flexible, efficient, scalable, portable, and modular way to program network devices [2,38], data plane programmability significantly reduces the time and effort required to configure otherwise tightly coupled network devices while offering network operators the flexibility of writing purpose-built functions.

In addition to moving away from using fixed-function networking hardware and adopting the SDN and NFV paradigms, data plane programmability also takes it a step further by enabling the support for defining stateless functions that can perform per packet operations while remaining target-agnostic. Programmability enables network operators to manage and automate the configuration of their networks easily; thus, it can improve scalability by automating repetitive packet processing tasks and processes. Additionally, it is instrumental in dynamic network environments, where network configurations change frequently – a commonplace in next-generation networks.

2.2.4 Network Function virtualisation

Using proprietary network hardware (e.g., routers, switches, firewalls, etc.) is expensive for service providers regarding procurement, security, configuration management, scalability,

and maintenance costs. Tightly coupled hardware middleboxes offer network operators less flexibility in implementing new functionalities to meet subscribers' demands [45]. NFV was introduced to solve the aforementioned challenges with traditional hardware middleboxes.

The European Telecommunications Standards Institute (ETSI) [46] introduced a high-level NFV architectural framework, envisaging the deployment of network functions as software running on the network function virtual infrastructure (NFVI), which could be a general-purpose server. This proposal was introduced to use hardware virtualisation [47, 48]. NFV has greatly simplified the deployment of network services because the cost of acquiring new hardware middleboxes is reduced, and several middleboxes can be virtualised and deployed on single or multiple general-purpose servers [10].

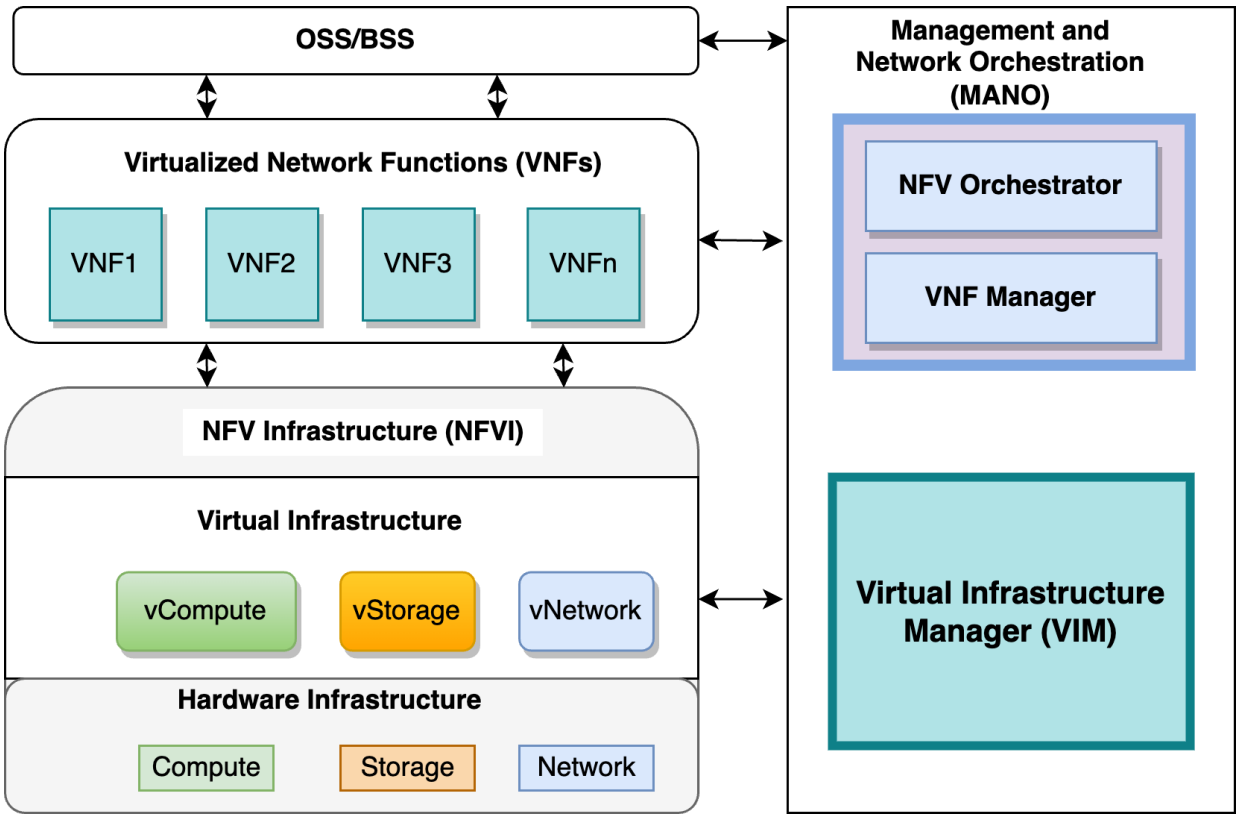

Figure 2.4: The ETSI NFV Architecture

The ETSI architectural framework for NFV depicted in Figure 2.4 shows all the essential components necessary for deploying NFV. The key components are briefly explained below:

- The operations support system (OSS) and business support system (BSS) directly interact with the vNFs. OSS typically handles tasks such as security and fault management, performance monitoring, and service provisioning [45]. OSS systems work with the NFV Management and Orchestration (MANO) to provide efficient and reliable operation of the NFV environment. BSS is concerned with tasks that involve monetizing

the services provided by the vNFs, such as billing, meeting SLAs, customer satisfaction etc.

- The management and network orchestration (MANO) component orchestrates vNFs lifecycle and chaining services in a scenario where service function chaining is used [3]. The functional blocks of this component, as seen in Figure 2.4, are the NFVO, VIM and VNFM. Note that the VIM manages the NFVI resources, and NFVO handles the orchestration of the vNFs.

- The vNF component is the network functionality, for example, a traffic load balancer, NAT module, WAN optimizer, an ACL firewall, etc. (Table 2.1), that has been virtualised to be implemented on general-purpose hardware, i.e., the NFVI. With the advancements in data plane programmability, programmable hardware devices such as FPGAs and programmable switches can also be used as part of the NFVI in NFV deployments [49].

- The hardware infrastructure consists of a virtual infrastructure with virtual computing, storage, and network components, which make up the NFVI. This infrastructure is managed by the virtual infrastructure manager (VIM), responsible for resource allocation and embedding virtual network functions on the virtual infrastructure.

- NFV Orchestrator (NFVO), which is an integral part of the ETSI NFV framework [46], is responsible for service orchestration and management [50]. One of the orchestration layer's functions is mapping virtual network functions in a service chain to available physical resources. NFVO coordinates the VNFM and VIM interaction to manage and allocate vNF instances in the NFV network environment.

vNF	Functionality
Application Gateway	Layer 7 traffic management based on application profile
DDoS Mitigation	Distributed Denial of Service mitigation and prevention
Layer 2 forwarder	Packet forwarding based on layer 2 information
Protocol Analyzer	Packet classification, based on the protocol in use
Flow tracker	Storing, displaying and forwarding ingress flows
Layer 3 Switch/Router	Traffic routing and switching using IP addresses
Application Firewall	Layer 3 and layer 7 packet filtering
ACL	User and application level access control
Bridge	Bridging between two networks or host devices
Carrier Grade NAT	IP address translation for WAN connectivity
Encryption gateway	Packet encapsulation and packet encryption/decryption
IDS/IPS	Stateful or stateless intrusion detection and prevention
Protocol converter	Protocol translation between IPv4 to IPv6
Protocol Accelerators	Performance improvements by ISPs
VLAN manager	VLAN encapsulation and decapsulation

Table 2.1: Commonly used virtual network functions [3]

- **NFV and data plane programmability**

  The proposed ETSI NFV architecture presents a framework that supports the virtualisation of hardware middleboxes (e.g., firewalls, load balancers, NAT, etc.), mainly using commodity servers as the hardware infrastructure to host virtual compute, storage and network [51].

  Leveraging the concept of data plane programmability, rather than staying restricted to using only commodity servers as the NFVI, network operators can incorporate programmable hardware switches or FPGAs into the NFV architecture, which can be programmed to run middlebox functionalities such as security services, the chaining measurement of network functions at the programmable data plane as in Dapper [52]. Typical central office functions can also be implemented on programmable hardware switches to provide subscriber-tailored services [53].

  Another solution that leverages the capability offered by the P4-based FPGA is the work by Ricart-Sanchez *et al.* [49], i.e., a 5G data plane designed mainly for achieving edge-to-core communication using FPGAs. Other efforts include the work by Singh

et al. [54] on offloading the virtual Evolved Packet Gateway (vEPG) of 4G networks onto a P4 programmable switch, where results showed low jitter and promising low latency of less than 2 μs.

The implementation of high-performance vNF offloading using hardware accelerators is being considered by network operators due to some key benefits, e.g., the reduction in energy utilisation and the number of hardware devices needed [55], and a reasonable reduction in packet processing latency [53]. P4-based FPGA hardware for implementing virtual network functions is also gradually becoming popular in service provider environments to achieve high performance and flexibility of vNF configuration [42,49]. A combination of programmable hardware platforms can be programmed to offer in-network services, e.g., DDoS mitigation [39], NAT [56] etc.

2.3 The Chaining of Virtual Network Functions

The vNFs implemented in NFV constitute the NF forwarding graph, which consists of network functions connected via logical network links to achieve packet processing by the vNFs. SFC comprises NFs connected in a chain (based on service requirements and specifications) to deliver end-to-end services to end users [46,57]. A typical service chain consists of NFs such as a NAT function, a firewall, and a traffic load balancer. Components such as the service classifier, SFF, Service Function Path, SFC proxy, and service function need to be in place for an SFC deployment to be complete.

Intelligent service orchestration is essential when handling various service functions, and this can be achieved when the NFV is adequately integrated with SDN [58]. In a typical SFC scenario, a classification of the user traffic is carried out by the flow classifier, which helps in deciding what network function(s) need to be traversed by the traffic before reaching its destination. When service requests traverse multiple NFs, a service orchestrator creates a chain of NFs that forms the final processing pipeline toward the destination (requested service) [3].

As more network operators continue to adopt network slicing, which serves as the enabler for next-generation networks, the chaining of virtual network functions for efficient service delivery has become commonplace due to the flexibility it offers network operators to deliver targeted services. Some available typical environments are the Gi-LAN network used by mobile network operators, residential/consumer services, and inter/intra-data-centre networks. Mobile network operators implement functions such as traffic optimizers, firewalls, carrier-grade network address translation (NAT), load balancers, and DPI at the core of the network, which is designed for subscribers that access Internet-based services [59].

- **Service chaining in 5G and Beyond Networks**

  Functions chaining is a key enabler of Beyond 5G networks [60,61]. Since 5G-enabled networks are characterized by low latency, programmability, and the support for diverse use-cases of the future, technologies such as NFV can allow providers to deploy services that are suitable for radio access networks (RANs) and mobile core networks [60]. By implication, using NFV, SDN, and SFC, service providers can easily provide tailored solutions that meet customer demands by carefully orchestrating user-generated traffic between an ordered list of network functions. The chaining of virtual network functions in SFC enables use cases such as the Gi-LAN mobile core network, residential and customer services, and inter and intra-datacentre networks [3]. Other important use cases such as self-driving cars, e-healthcare [62], and mixed reality (MR) and 5G-enabled IoT [62,63] are also possible due to the flexibility offered by 5G network slicing.

  Efforts such as the work by Morocho et al. [64] focus on showcasing how machine learning (ML) can be used to leverage the benefits provided by Beyond 5G networks. This work shows that ML can be used with enhanced mobile broadband (eMBB) and support future Beyond 5G applications that are envisaged to have high data rate requirements. Massive machine-type communications (mMTC) and ultra-reliable low-latency communications (URLLC) are also required to provide support for future use cases for Beyond 5G networks [65,66]. Abdelwahab et al. [67] explored how the 5G RAN can be enhanced using NFV, which could also lead to a reduction in the overall capital expenditure for telecommunications service providers (TSPs). As detailed in [67], some challenges that are related to 5G networks, such as efficient scalability of vNFs between physical networks, vNF performance guarantees, and simultaneously supporting the deployment of hardware and virtualised network functions, can be overcome with the flexibility offered by NFV implementations.

- **Function chaining in Inter-and Intra-Data-Centre Networks** SFC in the inter/intra DC environment allows for the chaining of virtualised enterprise network applications (in the case of intra-data-centre) and chaining across multiple locations, or inter-cloud, in the case of inter-data-centre networking. The ability of service functions to be instantiated across multiple data centres (inter-data-centre networking) is a crucial requirement for live VM migration. The SFC architecture should be designed to dynamically migrate service functions from one VM/container to another without disrupting user service requests [68].

  Deploying and managing service function chains in an inter-data-centre setting incurs bandwidth, deployment, intra-data-centre, and vNF costs [69]. In these environments, NFs are also used for policy-based routing of cloud services and enterprise

applications [59], which means that service providers can up-sell their services easily using SFC at the enterprise. This can be achieved by making network services user-programmable. Because NFV/SDN provides highly specialised solutions to meet customers' quality of service requirements, service providers can steer residential traffic, such as parental control and VoIP-related services, using service function chaining. The idea of follow-the-user service deployment in residential environments is a critical use case that is achieved using SFC implementations.

Network operators use Virtual Customer Premises Equipments (vCPEs) to easily create a service chain that meets user requirements [70]. Users in these environments are likelier to use web-based applications that use HTTP as the de facto protocol [71]. One of the requirements for such deployments is to create a service chain that prioritises security [72]. A typical example is a service chain that follows the order IDS > proxy > firewall.

2.4 NFV Deployments and Performance – A taxonomy of frameworks

This thesis section presents the implementation frameworks proposed for NFV/SFC deployments. A high-level taxonomy of NFV frameworks is presented after an in-depth review of the state-of-the-art NFV/SFC deployment frameworks [3]. The implementations reviewed set out to solve specific problems in the SFC domain, such as resource allocation and service orchestration, performance tuning, resilience, and fault recovery. Figure 2.5 summarises the taxonomy of the frameworks presented. Each classified framework strives to achieve diverse objectives and is evaluated against different baselines. Therefore, a taxonomy of the surveyed frameworks was created to compare the different works in their contexts quantitatively and qualitatively [3].

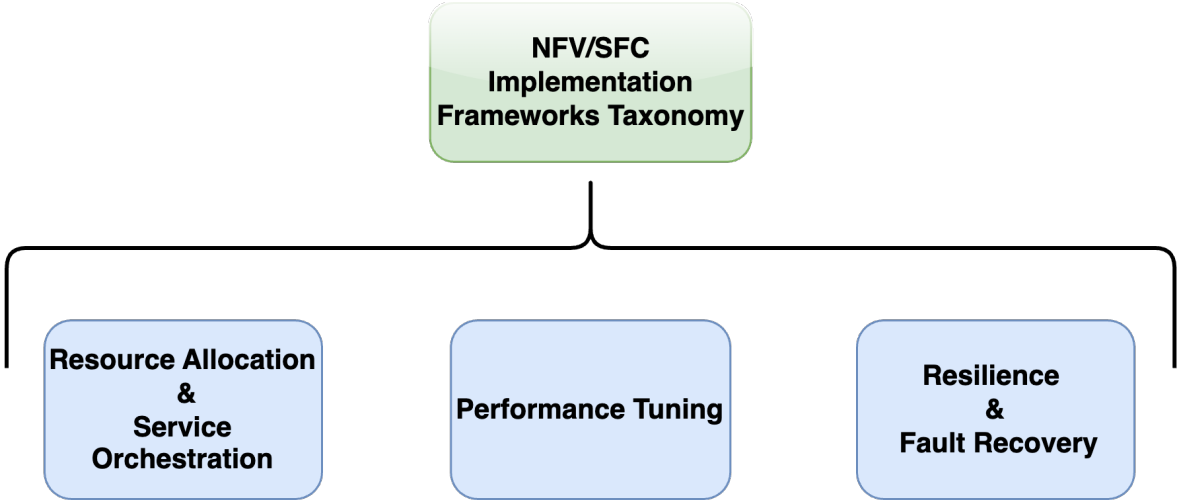

Figure 2.5: NFV/SFC frameworks taxonomy [3].

- **Resource Allocation and Service Orchestration** Efficiently allocating network, storage, and processing resources in virtualised network environments is critical in service provider networks. This is particularly true in NFV/SFC environments, where user traffic type and frequency can either be deterministic or nondeterministic, bringing the need for the underlying system to provide and allocate resources efficiently.

  The chaining of network functions to provide end-to-end services cannot be achieved without an efficient service orchestration scheme in place [73]. Efforts such as the work by Sun et al. [74] propose algorithms to handle SFC orchestration, resource utilization, and optimization. For example, the algorithm proposed by [74] is based on the breadth-first search (BFS) algorithm, which reduces overall chain-wide latency and bandwidth consumption. This category of the NFV/SFC taxonomy presents frameworks that have been designed to achieve the goal of efficiently allocating resources in NFV/SFC and orchestrating network services.

  Resource allocation and service orchestration frameworks [75–80] in general deal with the efficient utilization of available resources by employing techniques such as synthesizing packet processing graphs and offloading packet processing tasks onto smart NICs while ensuring that traffic is steered to the right network functions in a service chain and efficiently managing the life-cycle of network functions.

- **NFV performance tuning and improvements**

  Improving the performance of virtual network functions, either as standalone functions or as part of a service function chain, helps network operators reduce service instantiation costs [77] and the optimal use of available resources. This category considers frameworks that focus on optimising the performance of network functions in a service chain. Most frameworks designed to optimise performance devise mechanisms

that can achieve goals such as reducing network function deployment and provisioning time, maximising the throughput of network applications and reducing latency. Several efforts in the literature focus on optimising network functions using diverse technologies and implementation methodologies [3].

The frameworks under the performance-tuning category of the taxonomy [16, 18, 26, 81–87] are concerned with improving the overall performance of SFCs by employing techniques such as modular SFC deployments, lightweight packet processing elements, acceleration frameworks for packet processing, and deep learning techniques to improve the overall chain-wide scheduling, placements and performance. Several efforts in the literature aim to improve the performance of NFV deployments to meet defined SLA and improve overall QoS for subscribed users.

HyperNF is a high-performance NFV platform proposed by Yasukata et al. [86], which aims to properly utilise commodity server resources while scaling the number of network functions hosted by servers. The problem space addressed by HyperNF includes resource allocation, efficient utilisation, and high throughput when using everyday commodity servers to deploy virtual network functions. The proposed framework is aimed at large NF deployments, where utilisation is maximised for better throughput. Hypervisor-based I/O is employed, which helps reduce synchronisation overhead. HyperNF was designed using three core design objectives: (i) CPU cores are not reserved entirely for virtual I/O operations, thus providing high flexibility in terms of utilisation; (ii) proper accountability for virtual I/O tasks on respective VMs, thus offering a cohesive resource allocation strategy, and (iii) VM switches should not be used for packet switching; instead, the data path of software switches is exported to the hypervisor for forwarding and switching of packets. HyperNF was evaluated for scenarios involving a baseline setup using the VALE [88] switch for inter-VM communication, with each VM tied to a single CPU core, and second, a scenario that consolidates network functions in a shared CPU environment by varying the number of VMs (CPU cores are shared among the firewall VMs deployed using a round-robin scheme). Both scenarios outperformed the split and merge schemes compared with HyperNF. Other tests include resource allocation, NFV throughput, and SFC chain composition. A chain of 50 NFs can achieve a delay as low as 2 ms, which makes the framework ideal for SFC deployments.

Panda et al. [82] presented Netbricks, a platform for building and running virtual network functions that provide software isolation between NFs. The NetBricks framework differs from other approaches by (1) limiting the set of processing modules to core functionalities, which helps to reduce the number of modules that network application developers must deal with, and (2) allowing the customization of modules using user-defined functions, which makes the modules more flexible and optimised

for better network function(s) performance. NetBricks eliminates overheads resulting from context-switching by enforcing memory-level isolation in software and reducing I/O-related overheads by introducing zero-copy software isolation [82]. To evaluate the performance of the NetBricks framework, two example network functions were used: the first is a simple network function that decrements the TTL of a packet and discards any packet that has a TTL of 0, and the second is a stripped-down implementation of the Maglev load balancer [88], which splits ingress traffic among servers and also provides failure recovery for back-end servers. The measurements evaluated include simple NF overheads, array bound overheads, and how *general* the NetBricks programming abstractions can be. For the latter part of the evaluations, the programming abstractions, five network functions were implemented: NAT, firewall, Maglev load balancer, and a Snort-like NF that performs signature matching on ingress packets. The improved performance was observed for scenarios where (1) CPU cores and chain lengths were varied, (2) the load was varied concerning CPU cycles and chain length, and (3) throughput measurements for single network functions with a variable number of CPU cycles for multiple isolation approaches.

The NFV platform proposed by Zheng et al. [87], i.e., the Multiple Virtual Middlebox Platform (MVMP), is a high-performance framework that has been built using the Intel DPDK platform and Docker containers. The three major components of the proposed MVMP framework are (i) abstracted virtual devices, (ii) a control plane, and (iii) a shared memory space. Packet processing by NFs is achieved by an abstraction layer that supports the deployment of multiple NFs on a single hypervisor. Network functions are run in user space as processes, which makes them lightweight, thus requiring fewer resources for packet processing. Packets are polled directly from the NIC using the DPDK poll mode driver and sent to several network functions, adding to the proposed framework's fast packet processing speed.

In terms of implementation and evaluation, network functions were implemented and chained together. The service chain performance was evaluated and compared with the OpenNetVm [16] framework, which yields 3x better throughput as the number of network functions is increased in the chain, with an overhead of approximately 4% about network function isolation.

- **NFV Resilience and Fault Recovery**

  The resilience of virtual network functions to link, node and chain-wide failures has been addressed using diverse technologies and methodologies in the literature. The state of all network functions (active and standby) in a chain is vital when creating a resilience mechanism; thus, building a fault-tolerant middlebox and service chain becomes imperative [89]. Different frameworks use various mechanisms to detect faults,

fix them, and resume normal packet processing operations with as little downtime as possible [90]. The frameworks presented in [36, 89, 91–95] try to solve the problem of fault tolerance in SFC. These frameworks employ techniques such as network function replication and piggybacking of NF state changes across service chains to achieve resilience in SFCs.

Achieving cost-effective resilience is still an open challenge in SFC implementation frameworks, and there are unanswered questions for future research, including how SFCs respond to failure conditions, such as links between SFFs. The failure of the virtual network functions themselves is an aspect that needs to be considered as a long-term vision that requires further research, that is, considering having redundant SFs along the SFP to handle failure scenarios.

Synthesising network processing graphs or service chains is another approach employed to achieve high availability in SFC and using a multi-path routing approach [96]. As Mirjalily et al. [97] indicate, simple traffic load balancers can help with dynamic traffic re-routing to alternate processing pipelines in SFC environments. However, an efficient load-balancing algorithm that tracks device states and available resources must be implemented beyond the basic round-robin algorithm commonly deployed in today's network environments.

Approaches such as the work by Ghaznavi et al. [89] try to provide SFC resilience and, at the same time, eliminate the need for NF replication to reduce overhead. This is achieved by collecting and piggybacking NF state changes as packets traverse the service chain; thus, the overhead is reduced because the entire network function is not replicated to a standby node. A key challenge with proposals that employ service replication is the amount of overhead incurred with redundant backup links, nodes, and service chains; thus, the efficient implementation of a high-availability failure mechanism is still a research challenge that requires further attention.

The outcome of the in-depth literature review that was carried out, which led to the creation of the taxonomy presented in §2.4 helped shape the work in the subsequent chapters of this thesis. For the resource allocation and service orchestration category, the work presented in Chapter 4 caters to the resource utilisation of VNF hosts, specifically data plane hosts, at the time of placement. The work presented in Chapter 3 fits into the performance tuning category of the proposed taxonomy, i.e., using packet acceleration frameworks and the network data plane to improve the performance of virtual network functions. The final category of the taxonomy, i.e., resilience and fault recovery is also captured in Chapter 4 of the thesis. The proposal of a dynamic hybrid placement model responds to changes in edge network environments, i.e., ensuring that changes to optimal placements are quickly recomputed to minimise downtime

and avoid reaching any set latency threshold violations set by network operators.

2.5 Leveraging the Edge Computing Paradigm for VNF Placement

The placement of virtual Network Functions (vNFs) affects the latency between users and the vNFs. Thus, a better placement design needs to be implemented for future networks, which prioritises reducing the adverse effects of performance change caused by users' "hop-by-hop" movement between different vNFs.

Shahjalal *et al.* [98] formulated a Multi-objective Integer Linear Programming (MILP) problem to handle the placement of VNFs in a hybrid infrastructure. Network functions are placed on edge and cloud servers in a hybrid manner. Their approach minimises service deployment cost and maximises QoE, using user budget as a critical constraint.

Some efforts in the literature also consider the placement of virtual network functions at the network edge and cloud data centres [99–102]. Both Chen *et al.* [100] and Iordache *et al.* [99] considered the chaining of virtual network functions to create SFCs, the former formulated an SFC deployment cost minimization problem as an ILP, and considered constraints such as hardware dependencies and location. At the same time, the latter proposed an ILP model for latency-optimal on-path allocation of VNF chains on physical servers designed to operate in edge network environments. They also proposed the minimal path deviation heuristic algorithm, which minimises the end-to-end latency.

DeepNFV is a lightweight NFV framework proposed by Li et al. [84], which was explicitly designed for edge network deployments to minimise the packet processing tasks at the core of the network by offloading to edge network functions. DeepNFV is built on the GNF framework [18], which uses lightweight docker containers to build network functions for the edge. The key components of the proposed framework are the deep learning models employed and the infrastructure layer, which handles the interaction between network links and devices.

Like the GNF framework [18], DeepNFV was built to support moving network processing elements as close to the data source as possible (edge computing). As a use case for the DeepNFV framework, the authors considered network traffic analysis functionality by generating basic images from network traffic. A traffic analysis-containerized network function was used to analyse and classify images using deep learning models.

To demonstrate the traffic analysis use case, the DeepNFV framework starts by splitting the received traffic into discrete components, which are stored as PCAP files, and the second step involves the modification of the packet headers to trim the header length or remove unimportant fields from the packets. The modified PCAP files are *cleaned* to remove duplicates before being converted into image data. The resulting images are processed by the CNN model and sent to the next network function in the chain for further action(s). The ability of the framework to classify images and the performance of the network functions at the edge of the network were evaluated, and improved performance in terms of precision and efficiency was recorded.

Carpio *et al.* [103] presented a hybrid placement model to place network functions on VMs and containers using the edge-cloud continuum. They proposed a MILP model and a heuristic solution with cost minimization objective functions for operational costs, third-party cloud vNF placement costs, and penalties for SLA violation.

In contrast to the related works on edge VNF placement presented above, the hybrid placement approach (presented in §4.3.2) places VNFs on heterogeneous hosts at the network's edge with the objective functions of minimizing edge-to-device latency and fulfilling service requests while considering application latency thresholds set by network operators. Unlike previous VNF placement efforts, this thesis explores the benefits of combining the available processing resources on user space hosts (e.g., VMs, Containers, process-based elements, etc.) and the fast-processing speed of data plane hosts (e.g., programmable switches, SmartNICs), where less complex network functions can reside as part of a service chain.

The hybrid VNF placement method employed in this thesis efficiently handles dynamic edge placement scenarios while simultaneously minimising the number of latency violations based on set service level agreements (SLAs) by network operators. Additionally, unlike previous work, the proposed hybrid VNF placement scheduler reduces the number of VNF migrations between heterogeneous hosts with latency gains resulting from fewer optimal placement re-computations.

Previous work assumes a flat network topology in terms of hosts' capabilities, i.e., with less consideration for the different underlying capabilities of diverse hosts (e.g., data plane devices or packet processing acceleration frameworks) regarding the type of network functions best suited for improved performance. This thesis considers a hybrid scenario where VNFs are placed based on their type and complexity (e.g., delay-sensitive/delay-tolerant, stateless/stateful applications, etc.) in user space and the network data plane using heterogeneous packet processing elements.

The placement approach presented in this thesis embraces the heterogeneity of packet processing elements (i.e., the VNF hosts); in particular, the work leverages the process-

ing speed of the data plane and available resources on user space hosts when placing virtual network functions. Using the VNF placement scheme presented in this thesis, network operators can improve service delivery by delegating simple, time-critical operations to the data plane, leaving only more complex stateful processing for user space VNFs.

2.6 Summary

This chapter explored the advances in some key drivers of next-generation networks, i.e., NFV, SFC, data plane programmability and edge vNF placements. The evolution of traditional SDN into data plane programmability was also explored, noting the key differences and why we need data plane programmability in designing scalable and resilient networks (§2.2.2). NFV §2.2.4 and SFC §2.3 were explored in detail, including how the chaining of virtual network functions is utilized in delivering tailored services by network operators.

In §2.4, a taxonomy of NFV frameworks was presented, followed by the most recent literature on NFV performance improvements. This section also presented the synergy between using commodity servers to host virtualised middleboxes in the traditional ETSI architecture and packet processing acceleration frameworks such as P4-based Net-FPGAs. The concept of how network operators can leverage the edge computing paradigm was presented in §2.5. The remainder of the thesis is shaped by the work presented in previous sections; thus, the thesis seeks to explore the following key hypotheses as the major direction – following the insights obtained from exploring the state-of-the-art in this domain.

- As presented in the in-depth literature review in previous sections, there are diverse (mostly process-based) frameworks with different capabilities for deploying virtual network functions; also, the network data plane is a suitable candidate for deploying virtual network functions. Thus, practically evaluating the performance of the network data plane as a viable option for implementing vNFs becomes necessary. The practical performance of other deployment options, such as kernel by-pass, containers, and commodity servers, can pave the way for moving away from the status quo of vNF deployments by network operators, i.e., leveraging the diversity of packet processing elements to improve service delivery.
- Emerging edge use cases come with requirements for network functions that are computationally intensive as well as functions that require fewer computational

resources. Also, because the user space packet processing elements are generally characterized by high latency, we need to explore novel options for ensuring that network operators efficiently handle delay-sensitive and delay-tolerant applications to allow for faster service delivery.

A hybrid vNF placement model can yield optimized results and reduce vNF deployment costs by efficiently placing network functions in the network data plane and on user space hosts to better utilize available resources and offer faster service delivery. By implication, packet processing throughput can be improved while minimizing the end-to-end path latency. We can improve performance by delegating simple, time-critical operations to the data plane and leaving only more complex stateful processing for userspace vNFs.

– In addition to optimally placing virtual network functions at the network edge, in a hybrid context, Heuristic hybrid placement solutions are required to meet the particular requirement of edge network users, which involves constantly changing latency conditions due to factors such as mobility and traffic rerouting - thus, resulting to reoccurring path recalculation and latency threshold violations.

Chapter 3

On the Performance Benefits of Heterogeneous Virtual Network Function Execution Frameworks

3.1 Overview

Chapter 3 presents the design requirements and considerations for deploying network functions in the network data plane alongside user space packet processing elements to achieve faster service delivery by network operators. The complexities of virtual network functions and how design choices could affect the overall performance of these functions are also presented.

NFV and data plane programmability come with the promise of offering ease of service deployment, including reduced capital expenditure, to leverage these paradigms fully. Network operators adopt the chaining of virtualised network functions [3, 104]. As discussed in §2.3, a typical service chain consists of diverse vNFs with peculiar requirements in terms of, e.g., bandwidth, CPU, latency, etc., thus making the use of a fixed set of packet processing elements inefficient for catering to emerging use cases in next-generation networks.

Designing an SFC involves concatenating various virtual network functions to achieve end-to-end service delivery and service providers' SLA requirements [10, 104]. To address the aforementioned limitations, this chapter presents the design considerations for practically evaluating network operators' performance of diverse packet processing elements for efficient vNF placement. The chapter explores the need for using heterogeneous packet processing frameworks in deploying SFCs and considerations for data plane packet processing. The underlying technologies are also discussed, and

the design of a real testbed is presented, as well as evaluation results and key recommendations for network operators.

3.1.1 The Need for Heterogeneous Packet Processing Elements

Network operators often face the key challenge of delivering tailored services to subscribed users while meeting defined SLAs and responding to emerging use cases. This makes carefully considering the order of a chain imperative [104]. The IETF SFC architecture presented by Halpern *et al.* [105] shows that the SFCs are either bidirectional or unidirectional, where packet processing is performed through an ordered list of service functions in a unidirectional scenario [105]. A bidirectional SFC scenario requires packet processing elements to be placed in both directions of the service chain. SFCs are deployed as network service graphs, with SFs placed carefully at different parts of the service chain [106].

Regarding implementation, the agreed SLA between subscribers and network operators for the aforementioned service is also tied to the ordering of the functions in the chain, i.e., ensuring that packets are first inspected for any form of intrusion, then performing a deep packet inspection before finally carrying out access control filtering, could be key in adhering to defined SLA. The order of NFs depends on the particular use case and service provider policy in place; for example, a security policy would typically have a service chain that would constitute functions such as a stateless firewall (FW), a stateful Intrusion Detection System (IDS) and Deep Packet Inspection (DPI) module, which are computationally intensive.

Implementing the aforementioned chain would typically be in the order IDS $>$ DPI $>$ ACL between a source and destination node. The underlying devices/technologies that would be used for hosting the chain of functions depend on 1) the order of the chain and 2) the types of vNFs that constitute the chain. By implication, to deploy the example security policy chain, a network operator would concatenate a diverse set of underlying packet processing elements in an order that suits the use case. It is worth noting that a network operator who is providing a different service to edge users, e.g., a video sharing service, would have an SFC with a different order and types of functions (content caching and simple authentication functions), which will, in turn, determine the types of underlying packet processing elements used.

It is imperative to consider the available options for deploying vNFs by network operators to meet the diverse requirements of functions that subscribed users request. Because network functions have diverse requirements, Table 3.1 presents a represen-

tative classification of commonly used network functions. This is due to the nature of operations (computation) performed by stateful network functions on packets traversing them, which involves active state management. On the other hand, network functions such as L2FwD, FW, and Caching will benefit from data plane deployments, where a low memory footprint can handle vNFs with less requirement for state management [38, 107].

VNF	Description	Compute-bound	I/O-bound
Network Address Translation (NAT)	Private to public IP translation and vice versa	✓	✗
DPI – IDS/IDP	Traffic logging and inspection, stateful/stateless security	✓	✗
Virtual Private Network	User traffic encryption and security	✓	✗
L2 Switch	Packet switching based on MAC addresses	✓	✗
WAN Optimizer	ISP traffic optimization for QoS	✓	✗
Router	Packet routing based on L3 details	✓	✗
Load Balancer	Traffic load balancing, based on application level policies	✗	✓
ACL/Firewall	Device or application level access control	✗	✓
Caching	Performance improvements for better QoS/QoE	✗	✓
IPv4/IPv6 proxy	IPv4 to IPv6 connections proxy	✗	✓
Header Classifier	Classification, based on IP header fields	✗	✓

Table 3.1: Virtual Network Functions Operations and Classification [10]

3.2 Heterogeneous VNF Frameworks

Although the traditional ETSI NFV architectural framework used VMs as the primary hosts for network functions (introduced in §2.2.4), more recently, network operators have been using other frameworks as part of the NVFI for deploying network functions. The use of containers [14], unikernels [15], DPDK [16], Click [75], FPGAs [49] and SmartNICs are gradually becoming commonplace in service provider network environments. This section explores the key differences between some of these frameworks. It also presents the design of two performance evaluation testbeds for practically evaluating the performance of heterogeneous vNF frameworks using representative network functions.

3.2.1 VNFs on Containers, Commodity Servers and Kernel-Bypass

Figure 3.1 depicts the significant differences between the commonly used packet processing frameworks, i.e., Virtual Machines, Containers and Kernel-Bypass options such as DPDK or Click. Next, the high-level details of these frameworks are presented.

- **Containerized Network Functions**
  As depicted in Figure 3.1, using virtual machines for vNF deployment comes

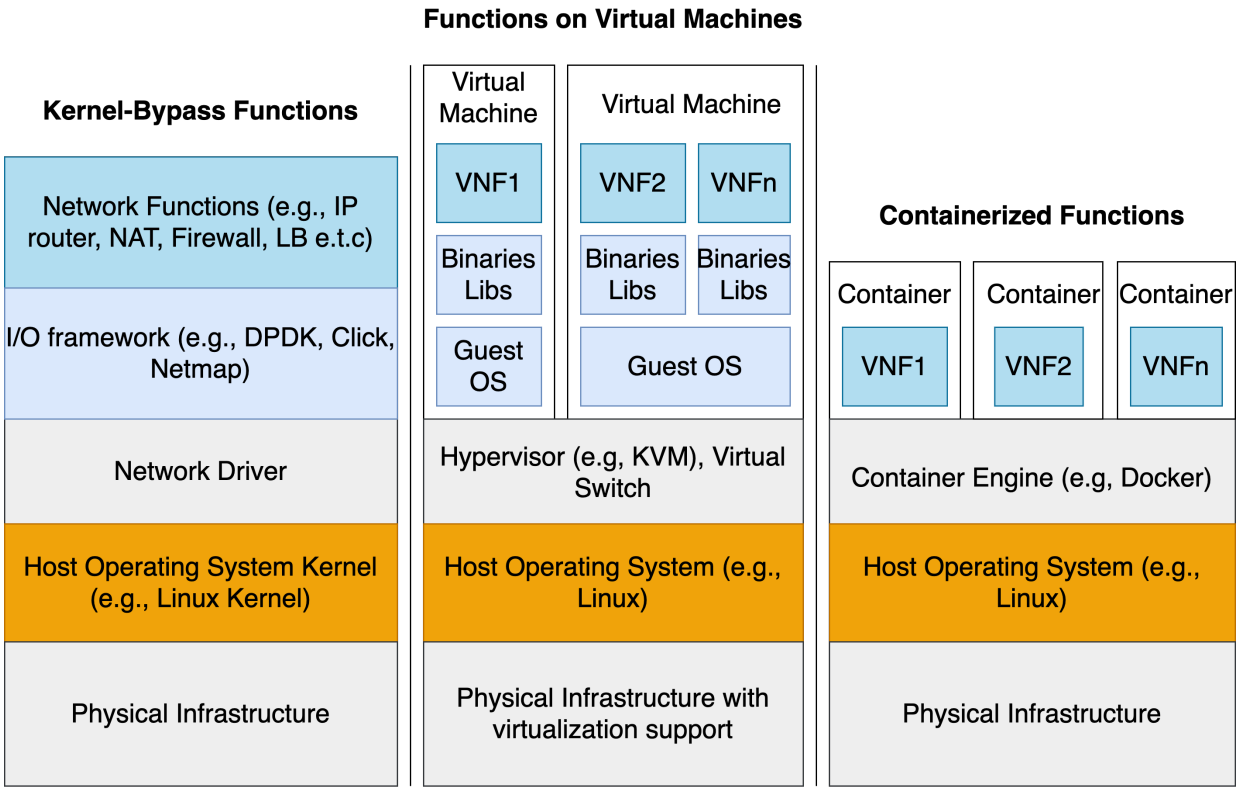

Figure 3.1: Virtual network functions on VMs, Containers and Kernel-Bypass frameworks

with the key advantage of allowing network operators to choose what Operating System they wish to use for each network function. In contrast, deployment options using kernel-bypass or containers must use specific modules, i.e., container engines such as Docker in the case of containerized vNF deployments or I/O frameworks, e.g., DPDK or Netmap in the case of kernel-bypass vNF implementations.

In terms of deployment overheads, VMs present more overhead than containers (20 – 30 % more [19]). Containers offer a high function-to-host density regarding the number of virtual network functions that can be hosted, thus reducing cost and footprint. The additional overhead introduced by VMs compared to containers is in terms of 1) high memory utilisation on the host, since VMs need to load a complete operating system on the host machine, 2) high CPU utilization due to operations such as memory management and system emulation, 3) overhead due to bootup time as the guest operating system is completely loaded at startup resulting to a slower startup process, and 4) additional overhead on the host due to storage requirements for guest operating system disk images needed for spinning up VMs [19].

Specialized software components or libraries typically perform packet processing tasks within a container. These components can leverage various techniques,

such as using raw sockets, interacting with network interfaces, or employing SDN frameworks to capture, analyze, modify, or forward network packets. By containerizing packet processing applications, the benefits of agility, resource efficiency, portability, scalability, and modularity offered by containers can be harnessed to streamline the deployment and management of packet processing functions within a more extensive network infrastructure.

The containerization of network functions also supports the micro-services deployment architecture, which allows small modular functions to be deployed based on demand – a preferred option for function deployments in most next-generation network environments.

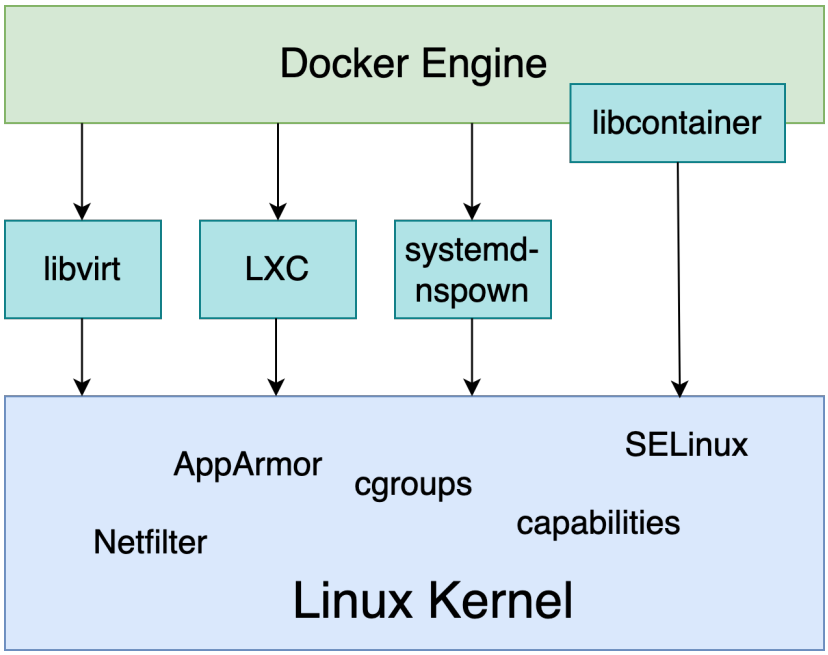

Figure 3.2: Docker and kernel libraries

The benefits above of using containerized network functions by network operators are not without any drawbacks; some of these drawbacks are briefly explained below:

* **Operating System and CPU Architecture Limitations**
  Because one of the requirements of running network functions as containers is the ability to share the operating system and kernel on the host device, the difference in the operating system and software architectures often makes creating containerized virtual network functions challenging. This also makes migrating vNFs between devices challenging, as network operators need to obtain prior knowledge of the operating system regarding compatibility, etc., before the migration of vNFs can occur efficiently. The difference in CPU architectures, e.g., x86 and ARM, also means that care-

ful consideration needs to be put in place to handle all dependencies while containerized functions.

* **Security and Isolation**
  In contrast to VMs or Kernel-bypass network functions, containerized functions do not handle isolation well due to the co-location of instances. Although this also improves the agility of containerized functions, interference between functions is possible. Efforts such as using AppArmor [108], a kernel security module for confining containers to a set of host resources, help provide security in containerized functions deployments. AppArmor provides application security by restricting devices, directories and file access using mandatory access control (MAC). Regarding resource isolation, control groups (groups) are designed to handle resource management, which helps to isolate resources such as memory, disk I/O, CPU, etc., in the Linux kernel [109].

– **Network Functions on Commodity Servers (Virtual Machines)**
  Unlike containers, virtual machines can completely substitute hardware machines. VMs encompass separate kernels and binaries and run as software on top of a hypervisor. By implication, each vNF configuration can be completely isolated in a virtual machine since the hypervisor can handle resource management between virtual machines. One of the key requirements in modern hypervisors is the support for hardware on the host CPU, which is supported by the most commonly used CPU architectures.

  Network operators aim to properly utilise commodity server resources while scaling the number of network functions hosted by servers. This includes solving challenges such as resource allocation and ensuring high packet throughput when using everyday commodity servers to deploy virtual network functions. Hypervisor-based I/O is employed, which helps reduce synchronisation overhead.

  Some other design objectives include: (i) CPU cores are not reserved entirely for virtual I/O operations, thus providing high flexibility in terms of utilisation; (ii) proper accountability for virtual I/O tasks on respective VMs, thus offering a cohesive resource allocation strategy, and (iii) VM switches should not be used for packet switching; instead, the data path of software switches is exported to the hypervisor for forwarding and switching of packets [86]. Single Root I/O Virtualisation (SR-IOV) improves functions' performance, enabling a single physical NIC to appear as multiple virtual interfaces, allowing VMs to access directly – PCIE passthrough [110].

– **DPDK Kernel-Bypass and Virtual Network Functions Offload**

In-kernel network stacks come with processing overheads [111,112] due to other operations handled by the kernel, e.g., scheduling, memory management, device driver operations, etc. The kernel needs to handle core functions and the intricacies of packet processing, leading to increased code complexity and potential for bugs and vulnerabilities [113]. To achieve high-speed packet I/O transfer in kernel-bypass frameworks, zero-copy reduces the I/O overhead associated with copying packets from the NIC for processing in the user space. Packets are DMA'd directly from the NIC to a shared memory space accessible to DPDK-based network functions. High-speed packet processing is also made possible by utilising the DPDK poll-mode driver rather than interrupts [16].

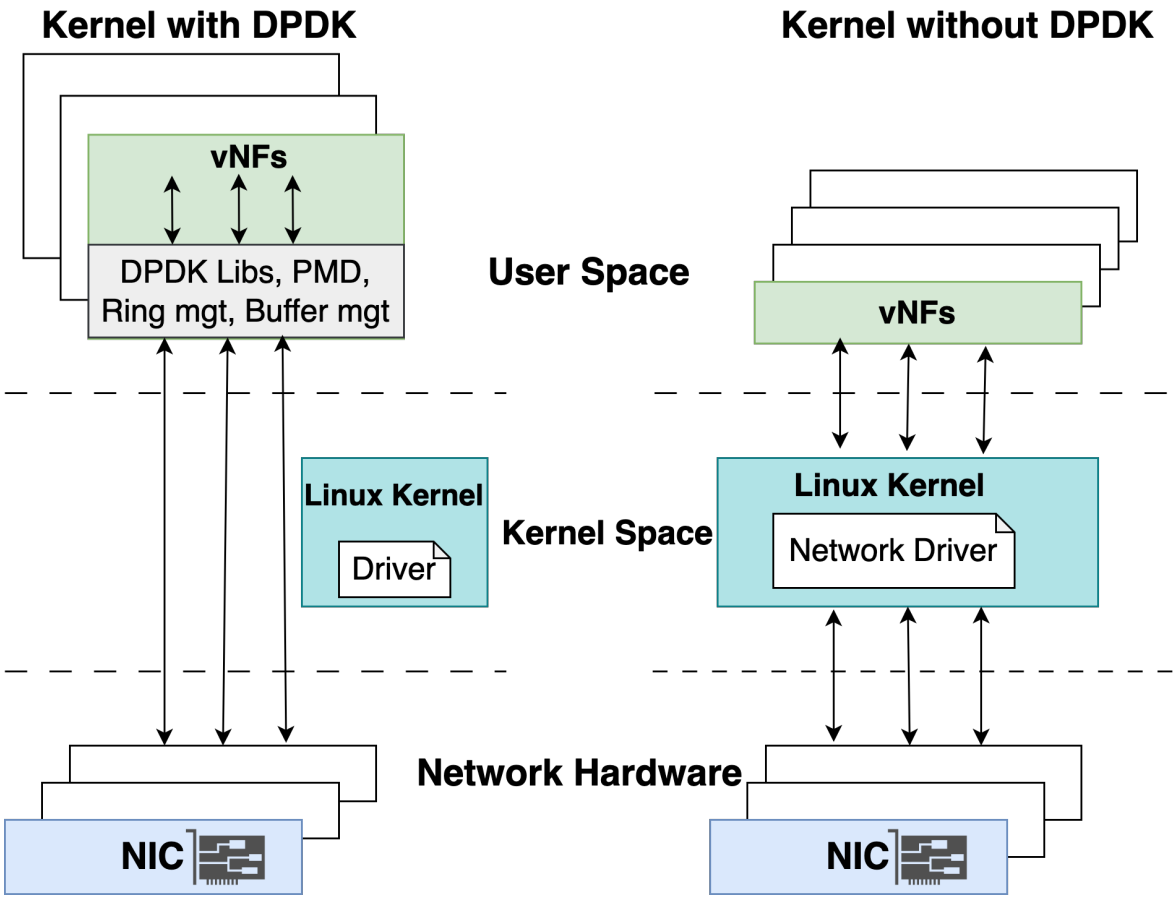

Figure 3.3: Network Functions using kernel-bypass with DPDK

Although the use of kernel-bypass frameworks such as DPDK comes with the promise of high throughput, one of the key design considerations is the high-CPU utilization often associated with DPDK-based network functions [10] since network functions are tied to CPU cores. Another consideration associated with DPDK-based network functions is the lack of in-situ support for basic kernel features such as ARP or basic TCP/IP functionalities – these need to be implemented by the network operator if required for the vNF to work.

Additionally, it is worth noting that DPDK offers performance enhancements in packet processing (presented later in the evaluations in §3.4.1).

Functions can also be implemented on top of SmartNICs, to relieve the operating system of the heavy lifting associated with packet processing tasks [79]. This approach can maximise throughput and achieve minimum latency by embedding network functions (eNFs) on in-network processors, reducing server CPU utilisation on end hosts.

Monoliths and Microservice Network Functions

Monolith and microservices application deployment has been adopted by network operators, with more operators leaning towards the microservices model of network functions deployments [114], which allows for the decomposition of network functions into small manageable components, unlike the monolith model, which offers tightly coupled application deployment [115]. In contrast to monolith application deployments, the microservices model offers some critical advantages to network operators, which include 1) ease of scalability, 2) ease of testing, 3) fault isolation and resilience of network functions, 4) simplified application development, and 5) ease of service upgrades. Although microservices NF deployment offers the aforementioned benefits, some challenges are faced, such as service decomposition and orchestration complexity [114].

The MicroNF framework proposed by Meng et al. [116] handles the deployment of modularised service chains, using a centralised controller for service chain graph reconstruction and redundant NF reuse. Network operators can easily describe the modularised chain deployed by clearly defining how the elements are interconnected. This is followed by processing the MSFC using graph reconstruction and identifying any dependencies by elements, aiming to reuse elements where possible. The reordered MSFC is optimally placed to reduce the latency between processing elements.

The primary goals of most modular SFC placement solutions are to (i) efficiently reuse elements that have similar configurations in the processing pipeline by first addressing the problem of dependency between different elements, (ii) solve the problem of VM-to-VM connection optimally using a virtual switch, (iii) shorten the service chain length, and reduce packet processing costs, where necessary. Regarding the scalability of NFs, run-time scaling algorithms are used, ensuring minimal inter-NF latency along the service chain. The problem of selecting processing elements ideal for consolidation is also handled by the MicroNF framework, in addition to a placement algorithm that prioritises high performance.

The framework by Meng et al. [83] has one of its primary goals to consolidate processing elements collocated on a VM and handle the placement of the modularized SFC to minimise packet transfer overhead between VMs. Using performance and resource-aware placement, the authors designed a placement scheme that selects SFC elements for consolidation. Fairness is achieved between several NFs tied to a single CPU core using a run-time scheduler implemented in CoCo.

In terms of scalability, the framework can utilise a push-aside scheme specifically designed to handle the reduction in performance, which might arise due to scaling elements. Unlike most existing NF scalability approaches that start up a VM when there is a need to scale, which leads to more overhead in terms of latency, the push-aside algorithm reduces the need for inter-VM hop creation. Rather than creating a new replica (as used by traditional vNF scalability solutions), additional processing resources are added to overloaded elements, resulting in more efficient resource management, one of the key benefits of adopting the microservices deployment model.

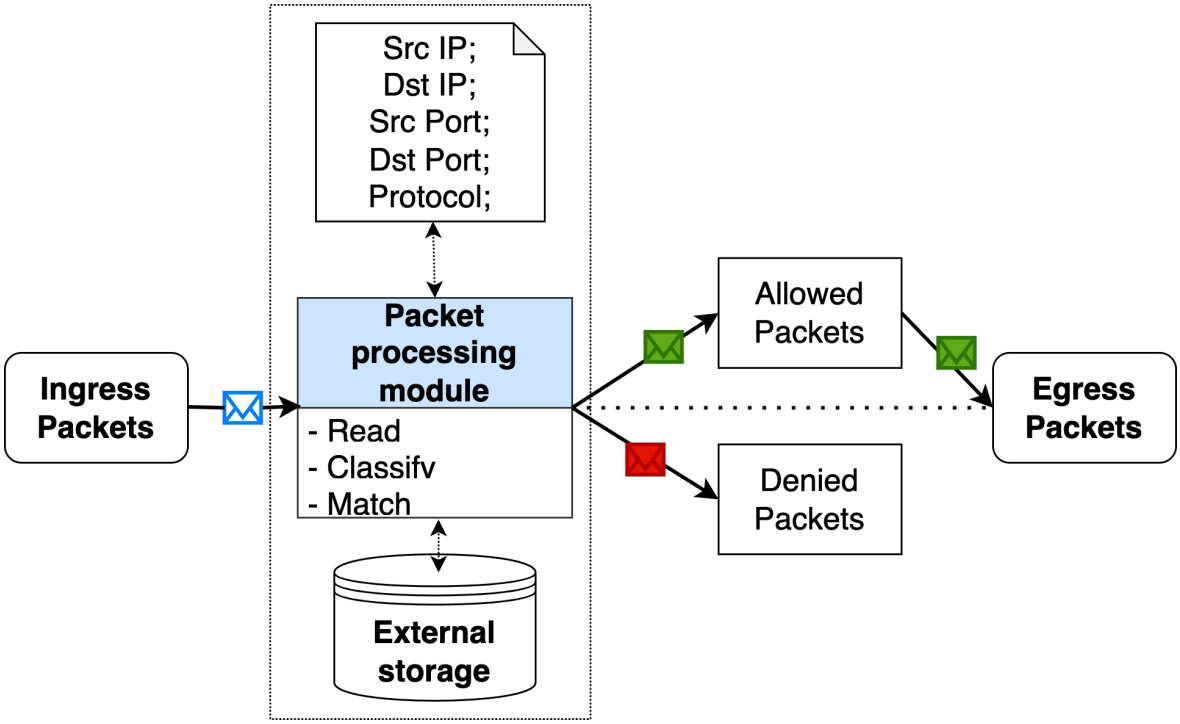

Figure 3.4: An overview of the operation of I/O-bound NFs, e.g., ACL/Firewall, Header Classifier, and caching. Tasks involve IP header lookup and data store access.

– **VNF Classification and Complexity** Table 3.1 presents some of the most commonly used NFs that can be abstracted and implemented in software and their classification. NFs are classified based on how they process packets and their typical operations along the processing pipeline. Figure 3.4 illustrates how packets traverse representative NFs that perform similar operations on received packets, such as an ACL/ Firewall. These NFs perform IP header lookups to read

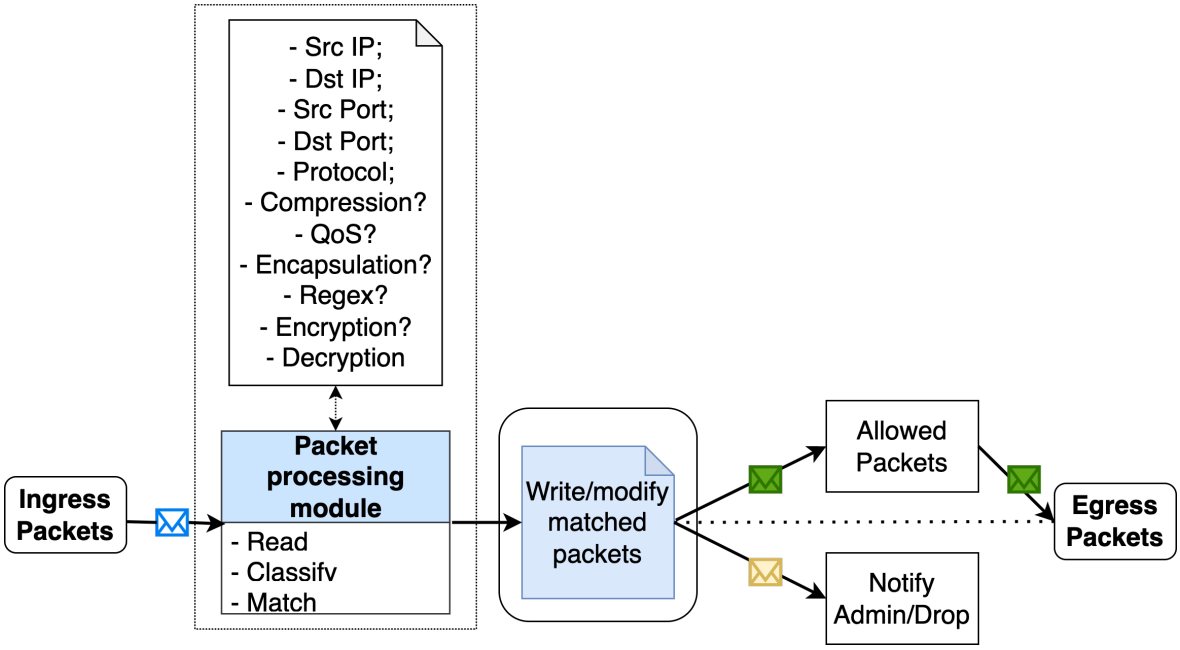

Figure 3.5: An overview of the operation of compute-bound NFs, e.g., NAT, IDS and QoS optimization. Tasks carried out involve more complex types of processing performed on packets.

fields such as source and destination addresses, port numbers, and the protocol in use. In the case of an ACL/Firewall, packets are checked against predefined match-action rules to either discard or forward them.

Figure 3.5 illustrates the typical mode of operation of the second group of representative NFs (Table 3.1), i.e., NFs that perform operations beyond basic packet header lookup against predefined match-action rules. Packets traversing these NFs go through more fine-grained inspection of payload data, depending on the type of NF. A typical operation might involve performing regex matches, compression parameters for Quality of Service optimizations or encryption parameters. Table 3.1 was deduced after carefully considering the behaviour of virtual network functions and frameworks that were implemented in notable works such as [3,117–119], where representative virtual network functions such as IDS/DPIs and L3 routers exhibit compute-bound characteristics. In contrast, others, such as ACL/firewalls, are I/O-bound due to the nature of operations they carry out on packets traversing them [75,120].

3.3 VNF Frameworks Performance Evaluation

This section presents the design components of the vNF frameworks performance evaluation testbed to practically evaluate the performance of some representative classes

of vNFs (presented in the classification in §3.2.1).

3.3.1 Data plane Packet Processing Considerations

Data plane packet processing refers to the handling and manipulating network packets within a networking device, such as a server, router or switch. This process involves various operations, including packet forwarding, filtering, routing, and traffic management. The data plane efficiently processes packets to ensure optimal network performance. However, data plane packet processing can be limited by factors such as packet size, the processing power of the device, and the complexity of packet manipulation operations [38].

One limitation of data plane packet processing is the processing power of the networking device. As network throughput increases, devices must process packets faster to meet the demand. However, a device's processing capability may become a bottleneck, leading to packet loss or delays. To overcome this limitation, network equipment manufacturers continuously work on developing more powerful processors and specialized hardware accelerators, such as FPGAs, SmartNICs, and programmable switches, to enhance the data plane packet processing capabilities.

Jumbo frames or large packets require more memory and processing resources, increasing the time and resources needed for packet manipulation. Moreover, devices along the network path may have different maximum packet sizes, leading to fragmentation and reassembly overhead. Efficient handling of jumbo frames requires careful consideration of memory management, buffer sizes, and processing optimizations when making data plane devices part of the packet processing pipeline.

The complexity of packet manipulation operations can also limit data plane packet processing. Security-related modules such as a DPI, encryption/decryption, and NAT involve more complex processing tasks that can impact the overall performance. These operations require additional processing time, memory, and computational resources, which can slow down packet processing and reduce the device's capacity [11]. Balancing the need for advanced packet manipulation features with performance requirements is crucial to ensure efficient data plane packet processing.

Device interoperability is another critical consideration in data plane packet processing. In a heterogeneous edge network environment, different devices from various vendors may have different programmability capabilities and interfaces. Ensuring interoperability and compatibility between programmable devices from other vendors can be challenging. Standardization efforts, such as open APIs and protocols, can help mitigate this limitation by promoting interoperability and ease of integration in

programmable data plane devices.

Regarding the implementation of the testbed used in this thesis, the ovs-vswitchd API was used through the ovs-ofctl command line tool, which simplifies the process of interacting and manipulating flow rules in Open vSwitch. The DPDK implementation on the testbed is built on the OpenNetVM framework [16], which uses the NFLib library as an API for communication between network functions and the NF manager. Figure 3.6 depicts the packet processing frameworks evaluated in this thesis section.

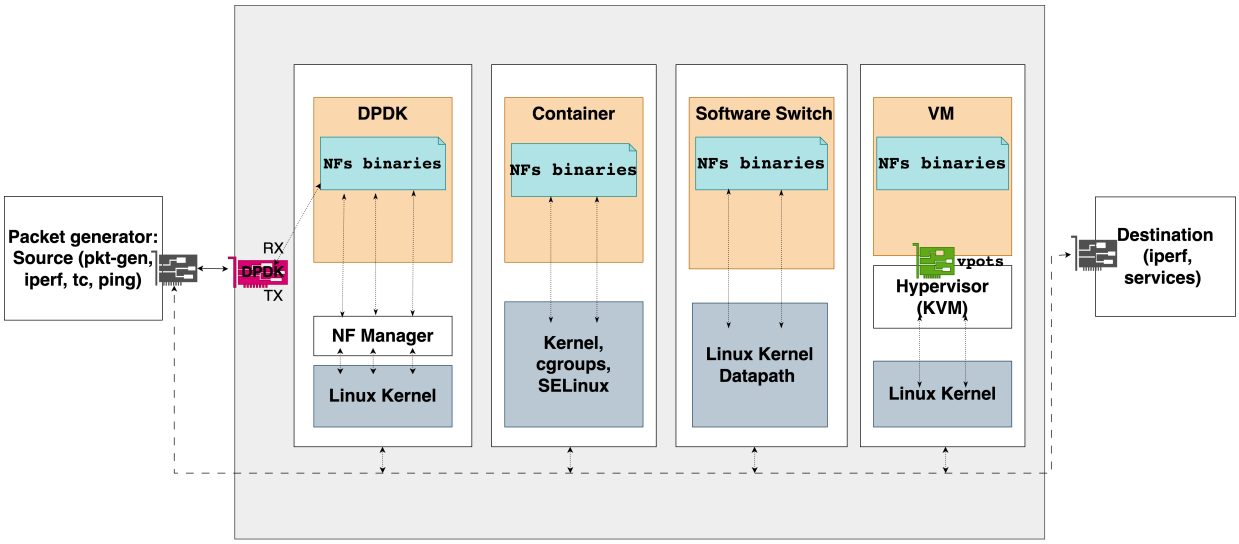

Figure 3.6: VNF frameworks performance evaluation testbed – packets are generated and made to traverse one or more diverse frameworks simultaneously. Match-action rules are written based on the type of functionality (e.g., a router or a firewall vNF).

3.3.2 Virtual Network Functions Testbed Implementations

Following the vNF classification presented in section 3.2.1, this section presents the implementation of representative network functions using heterogeneous VNF execution frameworks. One problem is finding out where to implement these functions best, which should allow for faster service deployments and for SFCs to be deployed using the suitable processing elements in the network.

3.3.2.1 Experimental setup

For performance reasons, hosting all VNFs (middleboxes) in a single server can lead to resource contention [121]. This informed the design decision to isolate the NFs from the packet generator (Figure 3.6) using multiple physical servers in the evaluation testbed. Investigations were carried out using physical servers connected back-to-back to eliminate overheads caused by switches.

To leverage the support of the Intel Dataplane Development Kit (DPDK), a DPDK-supported NIC is needed; thus, the Intel X-710 Dual-port SFP+ NICs were used, with speeds set to 1Gbps for consistency throughout the evaluations. The CPU on the Device Under Test is an Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, with six cores (without hyper-threading) for improved performance per core and 32GB memory. Linux kernel 5.4.0-59-generic on Ubuntu 20.04. A separate commodity server is a packet generator using the tools described in the traffic generation section. A second commodity server with identical specifications is Device Under Test (DUT), which has DPDK network functions from the OpenNetVM framework.

For the container scenario, each function runs as a separate instance for the containerised functions, leveraging Docker, which was installed on a separate commodity server with similar specifications as the DPDK scenario for performance consistency. Aside from their lightweight feature, container network functions (CNFs) allow for rapid deployment, scaling, and management of network functions. They are designed to be stateless and horizontally scalable, enabling efficient resource utilization and high availability.

The data plane component of SDN-aware virtual switches, such as the Open vSwitch that handles traffic forwarding, can be leveraged for performing packet processing tasks [30]. In the testbed presented in this thesis, Open-vSwitch was installed on the Device Under Test, with the programmed to emulate typical middlebox functionalities as firewall/access control and routing, creating the appropriate match-action tables [34]. The `ovs-vsctl` command line utility was used to add a network bridge to the virtual switch installation. TUNTAP interfaces were added and enabled for network connectivity, and both the physical and virtual interfaces were added to the Open-vSwitch bridge for communication between the host machine and the virtual switch function. Using the `ethtool` tool, the speeds on the virtual network ports were set to auto-negotiate mode and 1000 Mbps full duplex.

In Figure 3.6, VNFs leverage virtualisation technologies, such as the QEMU/KVM hypervisor, to create isolated VM instances that mimic the behaviour of dedicated hardware. The VNF binaries are installed within the VM, providing the necessary functionality, such as routing and firewall. Communication between VNF instances and external systems occurs over virtual network interfaces (vpot in Figure 3.6), allowing traffic routing between multiple network functions and physical networks. In the testbed, resources such as CPU, memory, and network interfaces were allocated from the resources on the host (hypervisor) machine. This enables scalability, as additional VNF instances can be deployed on-demand, and ensures isolation between VNFs to maintain security and performance.

Paravirtualised (virtio-net) NICs were used on the VMs for better performance. Packets destined for the HTTP and FTP servers were filtered to allow or drop based on protocol type, data link and transport layer protocols. This time, the same setup for the router NF tests was reused by creating match-action rules to route packets coming from the packet generator and destined for the FTP and HTTP service based on layer three information using LPM lookup. For a comparable scenario, the commodity server (DUT) hosting the network functions setup as the NFVI is the same server used for the software switch scenario. VMs running on Kernel-based Virtual Machine (KVM) with minimal versions of the Linux kernel 5.4.0-59-generic were used, with virtual network interfaces set to 1GB speeds.

Note that, in all the evaluation scenarios, in addition to ICMP packets aimed at sending packets to different subnets (in the routing function scenario), the destination nodes were set up with simple HTTP and FTP services to test the firewall functionality in terms of ports 20/21 and TCP ports 80. To allow traffic to pass through servers, the kernel IPv4 forwarding feature was enabled, making the routing functionality implementation possible.

3.3.2.2 Traffic generation and monitoring

The designed testbed used open-source tools for packet generation and measurements. Pktgen, meant for high-speed traffic generation to simulate and evaluate network performance, was used for IP packet generation.

In the context of this experiment, due to the Intel DPDK's high-speed line-rate packet processing capability, Layer 3 packets were generated, which allowed for the specification of source and destination addresses, including Time-to-Live and network(subnet) IP addresses. The choice of pktgen was also informed by its rich features, such as its flexibility for deciding packet rate, packet size, IP, and MAC addresses.

The kernel tc (traffic control) tool, which enables the configuration and scheduling of packets in the Linux kernel, was used and specifically used for shaping the traffic by setting qdisc (Queueing Discipline) on interfaces that needed to transfer packets at a defined rate, i.e., 1G in the switch data plane and KVM scenarios of the performance evaluations, which is in addition to the `ethtool` tool for setting the duplex mode, to full.

Iperf3 was used for bandwidth measurements in the testbed. This allowed us to specify TCP or UDP protocols and provide throughput values, including any losses or jitter (regarding changes to packets' arrival time). Packets are sent and made to traverse the network functions being evaluated, with bandwidth values being returned.

The ping protocol sends ICMP packets between source and destination through the tested network function for scenarios where access control or firewall functionality is evaluated. This was also used to test the routing functionality in the testbed, i.e., the ability of the function to route packets between different networks. Simple kernel tools, the top and the htop, were used to monitor CPU utilisation on the performance evaluation testbed, which helped identify any performance bottlenecks in the setup.

3.3.2.3 Network functions instantiation

- **Software switch functions**
  When functioning as a router, the Open-vSwitch switch matches the longest source and destination IP prefix and forwards packets using specified port numbers. The firewall functionality in the testbed matches IP addresses and specific TCP/UDP port numbers and denies or permits packets (based on defined match-action flow rules). For the firewall VNF configuration, match-action rules were provisioned on the virtual switch bridge to process packets destined for services, such as HTTP and FTP services, at any given time. Two separate flow tables were used in the Open-vSwitch scenario to implement the firewall and routing functions. Using `ovs-ofctl`, flows were added to the flow table, specifying source and destination ports and actions to be taken (*drop*, when denying access or *NORMAL*, which forwards received packets to egress interfaces). For the layer 3 routing rules (router function), source and destination IP addresses are used, with the type *0x0800* set, to indicate that the expected payload contains IPv4 packets, the time to live (TTL) is decremented at each hop, and the egress port/interface number is specified.
- **DPDK functions**
  In the testbed presented in Figure 3.6, the Data Plane Development Kit leverages the OpenNetVM NFV framework, which is ideal for building high-speed vNFs that can handle line-rate packet processing using zero-copy and the DPDK poll-mode driver, rather than interrupts [16]. The NF manager component of the testbed is built on the OpenNetVM manager. It represents the NFV MANO component responsible for NF life-cycle management. Functions were implemented as depicted in Figure 3.6; for the firewall NF scenario, the NF was initialised by creating an initial batch of 16,000 packets from the packet generator, which increases while the NF runs. The packets are returned to the RX queue of the generating NF (for measurements). This scenario was deployed to implement the firewall NF and then use the router Longest Prefix Match functionality on DPDK. The `firewall` and the `nfrouter` functions from OpenNetVM were

leveraged as the functions for this scenario.

- **Containerised functions**
  The firewall and router network function rules were added to lightweight Ubuntu base container images pulled from Docker Hub in this scenario. Similar rules, as in the Open-vSwitch scenario, were added to the instantiated containers in each scenario, i.e., firewall and router function, with iptables rules added. The virtual ethernet interfaces (Veth), which offer isolation by restricting container traffic within individual containers, are used for container networking between containerised functions and the host machine. In this scenario, the generated packets are made to traverse the containerised NF; match-action rules are inserted based on the type of functionality (a router or firewall function). For performance considerations, containers are instantiated and tied to CPU cores for a fair comparison. Kernel-based iptables firewall rules were used as the firewall function, denying/allowing traffic based on parameters such as source, destination IP, protocol and port number. Using the `iproute2` module, routes are added to the container, acting as the router function between the source and destination nodes.

- **KVM-based functions**
  The instantiation of functions in this scenario started with creating simple KVM virtual machines with two virtual CPU cores, 4GB RAM, and three virtual NICs (for communication with the host machine, source and destination nodes). Note that the storage requirement is insignificant while specifying the parameters of the virtual machine for the network functions, thus 20GB of storage was allocated to each VM at the time of creation. The kernel-based firewall function was configured using iptables to filter traffic sent to the destination node in figure 3.6, which is an implementation of equivalent functionality to the software switch firewall rules in terms of the operations carried out on received packets, i.e., matching packets based on source and destination ports or IP addresses. The router NF was configured as a simple Linux kernel-based router to route packets between two separate networks, with the packet generator and DUT server connected back to back (Figure 3.6). This was achieved using virtual machines solely created to host the given functionality (firewall and routing), with IPv4 forwarding enabled to allow traffic to pass through to the destination node on the testbed.

3.4 Heterogeneous Frameworks Evaluation

This section presents the results from the testbed described in the previous sections. Representative performance parameters such as throughput, packet rate and CPU utilization for each scenario are measured. In §3.4.1, the evaluation results of DPDK-based and containerised functions are presented, using commonly used representative virtual network functions in service provider network environments, i.e. a firewall and router, with focus on the million packets per seconds being processed by each function.

In §3.4.2, the evaluation results from the switch data plane functions are presented. Note that the bandwidth measurements in Figure 3.8 are averaged over ten runs each using `iperf3` (varying the `len` option to accommodate different packet sizes). For this scenario, bandwidth and CPU utilisation metrics were observed for each network function implementation. The results from the evaluation of KVM-based functions are presented in §3.4.3, detailing each representative network function's bandwidth and CPU utilisation.

3.4.1 DPDK and containerised functions performance

This section presents the findings using a framework that handles line-rate packet generation and processing. This helps discover the behaviour of commonly used representative network functions on a high-performance vNF execution framework and lightweight containers. The experiment was carried out using the Intel Data-Plane Development Kit (DPDK)[1] framework and Docker containers. The results are as shown in Figure 3.7, where the packets per second rates obtained are depicted for the minimum-sized (64B) and MTU-sized (1500B) packets. The DPDK firewall NF processed about 2Mpps, with the containerised firewall function processing about 0.7Mpps (only 35 % of packets sent) for 64B packets.

3.4.2 Software switch data plane functions performance

This section presents the results from the implementation of the software switch functions. The supported bandwidth achieved on the switch data-plane firewall scenario is 226Mbps for 64B packets. Meanwhile, the largest packets (1500B) were transferred at a bandwidth of 957Mbps, which is about 96% of the supported line rate. The switch data plane router NF processed 185Mbps at small packet sizes (64B) and 905Mbps for large ones (1500B). In addition to Figure 3.8, to put the results into perspective, Table

[1] https://www.dpdk.org/

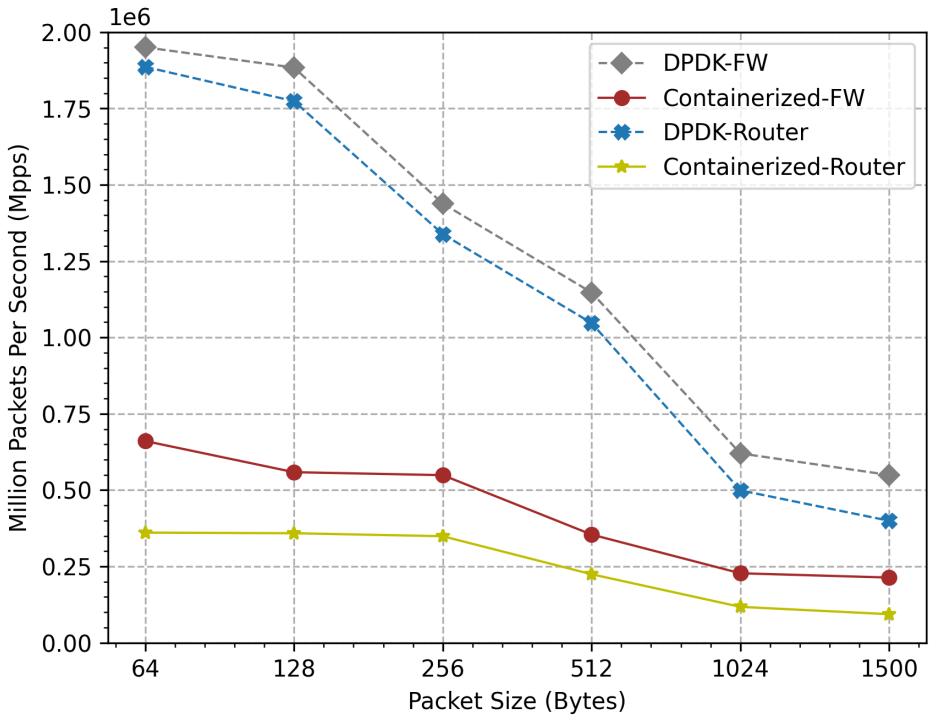

Figure 3.7: Million Packets Per Second – DPDK and Containerized vNFs. All NFs are tied to separate CPU cores for performance isolation and measurement consistency.

3.2 summarizes the bandwidth for the switch data plane and KVM-based functions implementations of the firewall and router NFs, with various packet sizes.

The switch data plane implementation scenario yielded about 14% more bandwidth for 1500B packets (Figure 3.8). NFs (router and firewall) produced a reasonably high bandwidth compared to the KVM-based function implementations. The drop in performance observed with small packet sizes is due to per-packet overheads and a higher required PPS, which is seen as a drawback in most software-based middleboxes. The CPU utilisation is presented in Figure 3.9, with the switch data plane firewall function utilising fewer resources (details in §3.4.3).

3.4.3 KVM-based functions performance

In this evaluation, similar to the switch data plane scenario, the performance while sending various packet sizes, with packets traversing the VM network functions, was measured and presented in Figure 3.8 and Table 3.2. With the firewall functionality in place and the packet generator attempting to gain access to the server hosting network services, the bandwidth peaked at 150Mbps (for 64B packets), a significant decrease from 226Mbps observed in the switch data plane scenario. Similar tests were carried

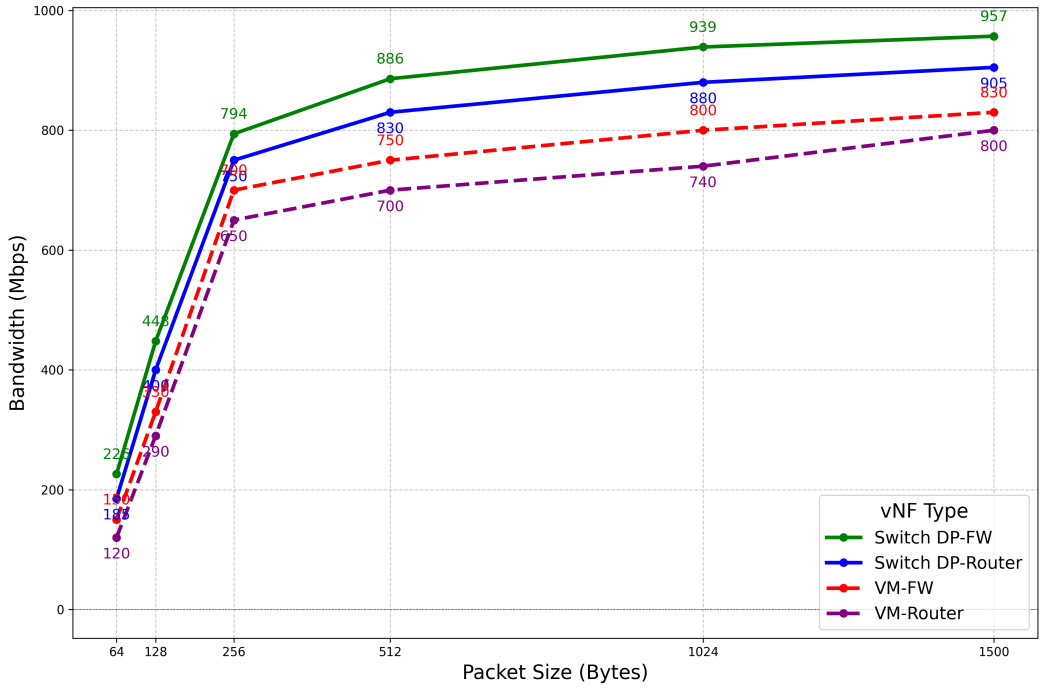

Figure 3.8: Mean bandwidth rates for switch data plane and KVM-based deployments of network functions.

Table 3.2: Mean bandwidth rates (in Mbps) for data-plane and VNF deployments.

Packet size (B)	DP-Firewall	VM-Firewall	DP-Router	VM-Router
64	226	150	185	120
128	448	330	400	290
256	794	700	750	650
512	886	750	830	700
1024	939	800	880	740
1500	957	830	905	800

out for the router NF, producing a bandwidth of 120Mbps for 64Bytes packets. MTU-sized packets produced 800Mbps, which is about 11% below the bandwidth reached in the corresponding data-plane router scenario, i.e., 905Mbps.

For MTU-sized packets, the bandwidth of 830Mbps (for the firewall NF scenario) was achieved, which is 13% less than the switch data plane firewall scenario. For this first set of experiments, it is observed that the switch data plane scenarios outperform the VM functions implementations regarding maximum bandwidth and CPU utilization; this also holds for the packets per second rates (Figure 3.8).

The CPU utilization of the commodity server hosting the testbed is depicted in Figure 3.9, which gives the processing cost of having the NFs deployed in both scenarios. By implication, regarding CPU utilization cost, the switch data plane implementation of the firewall requires fewer resources. CPU spikes were much higher in the firewall VM scenario (up to 120%, which is precisely 10% of the overall CPU cores on the server. Note that higher CPU utilization was achieved for smaller packet sizes, which decreases as packet size increases (Figure 3.9).

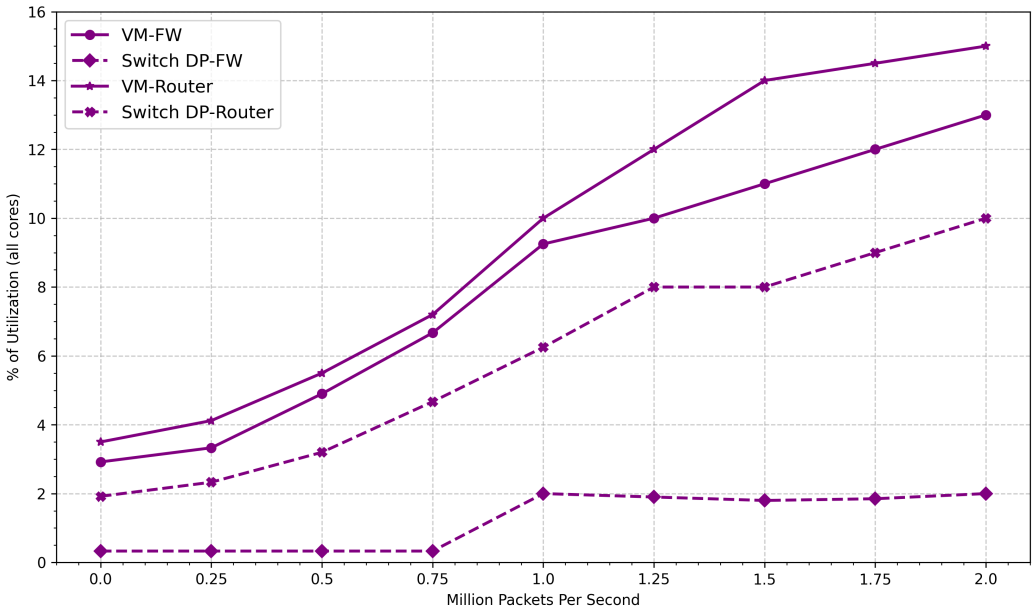

Figure 3.9: CPU utilization – switch data plane firewall and router maintain much lower CPU utilization when compared with VM functions, as the million packets per second increases.

3.4.4 VNF Suitability and Recommendations

Evaluations were carried out with multiple physical commodity servers to fully appreciate how deployment choices by network operators can affect NFV performance. Also, containerized and DPDK NF scenarios benefited from the design choice of run-

ning NFs on separate CPU cores. Another design consideration is the behaviour of vNFs built using kernel-bypass frameworks such as the Intel DPDK library. The number of available CPU cores is especially relevant when network operators adopt this approach. This is even more so as vNFs are often tied to available cores, and depending on the use case, some cores need to be made available for the NF manager or controller.

By integrating DPDK in the packet processing pipeline, functions can be executed directly on the physical host's hardware, bypassing the overhead of traditional hypervisor-based networking. DPDK allows VNFs to access and process packets directly from the host's network interface, drastically enhancing throughput and reducing latency. This integration enables the efficient deployment of network functions, such as firewalls, load balancers, and deep packet inspection, as lightweight and high-performance instances. As a result, DPDK-based VNFs achieve better scalability and responsiveness in modern virtualised network architectures [34, 122].

The performance of the data plane switch implementation is particularly noticeable with larger packet sizes, especially in the firewall VNF tests. This consistency also holds for CPU utilization. By implication, service providers can leverage the data plane of the network, especially in situations where compute-intensive NFs are deployed. The overall performance of network functions evaluated increased significantly using DPDK, even for 64B packets. Service providers can also leverage this to deploy compute-intensive vNFs, such as a stateful IDS, for faster packet processing.

The mode of operation of some network functions, such as WAN optimisers, involves carrying out tasks such as traffic compression/decompression, caching, floating-point operations and sometimes content duplication [78]. Current switch data plane technology does not readily support such operations; hence, service providers can leverage other heterogeneous deployment options.

One of the practical scenarios that would benefit from the results of the evaluations presented is a network operator that provides a set of network functions comprising a gateway firewall and an Intrusion Detection System (IDS), a scenario explained in §3.2. This chain of network functions currently being implemented in the user space of the network can be split in a hybrid manner between the data plane and user space to improve performance and reduce processing and service deployment costs.

A network function such as an IDS, which is stateful, requires security modules and is computationally intensive [119]. This will benefit from the available resources (memory, CPU and I/O) in the user space of commodity servers while having the firewall functionality or other lightweight functions deployed at the network data plane. Also, depending on application profiles and requirements, a hybrid deployment scheme can

be employed in some existing and emerging edge network use cases, such as e-healthcare, self-driving cars and mixed reality (MR). Simple, time-critical operations can be delegated to the data plane, leaving only more complex, stateful and resource-hungry processing for user-space VNFs.

3.5 Summary

This chapter presented the key design requirements and the need to use heterogeneous packet processing elements to deploy virtual network functions in service provider network environments. The justification for using heterogeneous packet processing elements was also presented in §3.1.1, including SFC structures, underlying packet processing technologies and classification of representative virtual network functions.

To assist network operators with the decision-making process of selecting the appropriate packet processing frameworks for creating a service chain, this chapter also presented the key design components of a vNF performance evaluation testbed, suitable for ascertaining the performance benefits of heterogeneous packet processing frameworks for NFV deployments in next-generation networks. In §3.2, the key technical differences between some commonly used packet processing frameworks were presented.

The design implements commonly used network functions (a firewall and a router) using the data plane component of a production-grade virtual switch and network functions deployed on separate virtualised processing elements §3.3. A high-performance scenario using the Intel DPDK framework and a scenario with lightweight containerised network functions were considered for deploying the same network functions, which shows a significant increase in the performance of the representative network functions considered. Results from the evaluations were presented in §3.4, including a detailed explanation of some recommendations to network operators in §3.4.4 regarding the considerations for implementing virtual network functions.

This chapter's design principles and evaluations will help network operators deploy diverse network functions on heterogeneous hosts. Depending on application profiles, NF complexity, and resource availability in the network infrastructure (presented in the next chapter), a hybrid deployment framework where service providers deploy NFs using the network data plane and user-space processing elements is feasible.

Chapter 4

Latency-aware VNF Placement on Heterogeneous Edge Hosts

4.1 Overview

Emerging edge computing use cases make finding innovative ways to implement virtual network functions in edge network environments imperative. It is now commonplace to have a chain of vNFs consisting of functions with diverse latency, bandwidth and resource requirements (Chapter 3). On the other end of the vNF deployment spectrum, diverse packet processing elements are available to network operators for function placements.

In addition to exploring the capabilities of diverse packet processing elements for edge vNF placement, there is also a need to shift from the status quo of vNF placement, i.e., by leveraging the processing capability of the programmable data plane and incorporating the same as part of the end-to-end packet processing pipeline. Although attempting to leverage the capability of the programmable data plane for the placement of vNFs has some promising results in terms of improved service delivery, it is worth noting that a subset of functions is not necessarily ideal for programmable data plane implementation due to their particular requirements, e.g., high storage, compute, and encryption/decryption.

This chapter presents the motivation for considering the network data plane for vNF placement and why a hybrid placement scheme is needed to meet the diverse demands of edge use cases in next-generation networks. It also presents the fundamentals for latency-aware hybrid placement of vNFs on heterogeneous hosts, which addresses the lack of frameworks that can leverage the network data plane to offer end-to-end programmability along a given network path in the edge infrastructure.

4.2 Design Considerations

This section presents some design considerations for deploying network functions on heterogeneous NFV frameworks, thus extending some of the considerations presented by the IETF draft on lightweight NFV technologies [123] and the ETSI NFV standard on VNF implementation [51].

4.2.1 The Need for Hybrid VNF Placements

There are diverse frameworks for implementing virtual network functions in service provider network environments, which can be used to implement network functions at the data plane of the network using virtual switching technologies such as Open-vSwitch [124] or P4 [9]. Virtual network functions can also be implemented in the user space, using unikernels [125], virtual machines [126], FPGA [127], or, containers [16] – or accelerated using frameworks such as eBPF [40, 128] or DPDK [92].

As detailed in §3.3.2, Open-vSwitch is one of the earliest and widely used frameworks that supports defining virtual network functions using a set of match-action rules. The OpenFlow protocol (§2.2.1 of Chapter 2) has seen commercial adoption in service provider networks since its first release [129], which also serves as an essential component for achieving control and data plane separation in SDN.

Because Open-vSwitch, using the OpenFlow protocol, does not inherently support stateful data plane processing [130], it becomes ideal for implementing simple stateless functionalities, e.g., a stateless firewall or router functionality, as demonstrated in the experiments presented in Chapter 3. Few exceptions are possible regarding achieving stateful packet processing using Open-vSwitch, e.g., the stateful firewall functionalities presented in [131, 132] and stateful NAT [133]. These are essentially achieved by extending Open-vSwitch using the `conntrack` kernel module or other custom-built APIs [134, 135] and are known to add additional overhead, which impacts performance [133].

The introduction of P4 [9], which adopts the Reconfigurable Match-Action Table (RMT) architecture for programming packet processing pipelines addressed some of the limitations of the Open-vSwitch data plane (see §2.2.3 in Chapter 2), which includes being device agnostic, providing an architecture that allows network engineers to flexibly program rules for specific targets, the ability to define a pipeline of tables that are only constrained by the available memory on the target device and support for dynamic match-action tables and more protocols for defining packet processing pipelines. One of the main breakthroughs of P4 has been the ability to perform (cus-

tom) real-time packet processing, which OpenFlow can only do through an SDN controller. This provides network operators a much more flexible option for programming functions that can be implemented on various targets for different use cases. Using P4, stateful packet processing is being achieved in some example use cases, such as the implementation of key-values stores as in [136] and in-network telemetry computations as in [23], whilst retaining performance.

Unikernels [125], specialised lightweight Virtual Machines, serve as an option for deploying user space network functions. Several factors have hindered the wide adoption of unikernels by network operators for deploying applications in the past, some of which are a lack of fundamental security features, a lack of Portable Operating System Interface (POSIX) compatibility, limited adoption of a tooling ecosystem, etc.

These limitations were addressed by introducing the Unikraft architecture [137], which provides POSIX compatibility, security features support, and a fully modular architecture directly translating to high performance [138]. ClickOS [81] is one notable implementation that uses unikernels for creating lightweight network functions based on the Click router [139]. The ability of unikernels to only use parts of the Operating System while running on special hypervisors increases their portability; thus, their specialised lightweight nature makes them ideal for serving real-time user space applications, as more recently demonstrated in [17].

Unlike unikernels, network functions deployed on Virtual Machines are implemented on virtual instances of a full-blown machine, i.e., encapsulating the functionality of an OS, libraries, a separate kernel, CPU, memory, configuration and binaries running on top of hypervisors such as KVM (see more details in §3.2.1 and §3.4.4 in the previous chapter). This makes VMs ideal for deploying compute-intensive user space functions that can easily leverage the available CPU, memory and storage the virtual environment offers. In contrast to Virtual Machines, containers (see §3.2.1) are lightweight options for deploying virtual network functions, which presents less processing overheads [18, 19]. Results from the experiments presented in the previous chapter also show the performance gains of using containerised functions (see §3.4.1).

FPGAs are also good candidates for deploying high-performance network functions, which are enabled by their support for ASIC-like performance, thus offering low latency and high throughput, including runtime reconfigurability [2]. FPGAs offer line rate packet processing and can readily be integrated with other acceleration frameworks, such as the Intel DPDK, as demonstrated in [24].

The wide adoption of FPGAs for deploying network functions has also birthed some FPGA-supported NICs, e.g., the Xilinx Alveo [140]. A recent use case example is eHDL [141], which serves as a synthesis tool for generating eBPF/XDP programs,

with the Xilinx Alveo U50 FPGA NIC as a target. Other concrete examples of functions implementation on FPGAs include Metron [77], ClickNP [142] and Flightplan [22].

The ability to capture and filter user-level network packets that matched defined rules was one of the initial design goals of the original BSD Packet Filter (BPF) – which also allowed the safe execution of user-written programs in kernel space [143]. The design employs a register-based machine easily supported by commonly used register-based Reduced Instruction Set Computer (RISC) processors. The availability of larger address spaces also allowed BPF to use a non-shared buffer model. BPF has since evolved with the introduction of the Extended Berkeley Packet Filter (eBPF) to better leverage modern processors' capabilities. Changes such as the use of Just-in-Time (JIT) compilation, increasing the number of registers to ten from two, and migrating to 64-bit registers have helped boost performance since this means, like native hardware, parameters can be passed directly to functions in eBPF virtual machine registers [144]. Kernel hooks like the eXpress Data Path (XDP) use eBPF to safely place user-written code into the packet processing pipeline [111]. This makes eBPF ideal for running functions that can safely be implemented within the kernel stack, as demonstrated in Galette [40] and eHDL [141].

In contrast to eBPF, to achieve fast packet processing, the Intel Data Plane Development Kit (DPDK) completely bypasses the kernel, using poll mode drivers which map NICs to transmit and receive queues to handle device operation and memory allocation (see §3.2.1 for more details). DPDK has seen wide adoption in implementing network functions that can achieve high performance by bypassing the kernel stack [3]. Some common examples using DPDK to accelerate packet processing speeds include OpenNetVM [16], FlexiIPS [25] and Phantom SFC [26]. DPDK has also been used to achieve ultra-low latency packet processing [27]. Some recent use cases, such as Middlenet [128], combine eBPF and DPDK's capabilities to achieve high packet processing performance.

Van et al. [145] proposed a hybrid NFV framework for building low-latency and high-throughput vNFs using XDP. The XDP program handles simple operations, whereas a user-space program handles complex operations. Three example NFs were chained (with two SFCs) using OpenStack. The proposed framework separates packet processing into fast and slow paths, with less complex NFs (simple LB and flow statistics) handled by the slow path, whereas vNFs that require complex processing are handled by the fast path. Another use case scenario is when functions are implemented in XDP using a simple hash algorithm and are implemented using a hybrid approach that combines user space and kernel space if the NF is sophisticated, e.g., [146–148].

Because NFV and data plane programmability allow for the creation of tailored network functions for efficient packet processing, it is ideal for creating a processing pipeline that supports fast service delivery and efficient use of available resources by service providers to support future networks. Creating a service function chain (SFC) currently involves chaining network functions that are either implemented at the data plane or as user-space functions and are often carried out using network functions built using the same framework to create processing elements. Current frameworks create SFCs through chaining a single type of vNFs; thus, very few try to chain components from different frameworks [40], let alone include programmable data plane processing elements as part of the chain.

The current data-plane technology is resource-constrained and does not easily support operations such as manipulating floating-point values [11]. This is partly because programmable data plane hardware was designed to achieve packet processing proficiency; general purpose NPU NICs are no exception [149]. To achieve online learning in the data plane, Opal [149] employed classical techniques of Reinforcement Learning, which leverages the inherent parallelism of SmartNICs to allow flow inference at line rate.

Thus, the resource-constrained nature of the data plane also poses some limitations in implementing resource-hungry network functions, e.g., network functions that require constant state management. In addition to being resource-constrained, the bound execution time also means the programmable data plane is not readily capable of handling turing-complete computations and loops [2].

We question how network functions are currently implemented (including the composition of the network functions that create a service chain) and seek to argue for a hybrid implementation approach that combines user space and data-plane components to create a chain of network functions along the processing pipeline.

It is correct to state that diverse frameworks have their constraints and benefits. A hybrid implementation framework will leverage user space and data plane benefits by composing processing pipelines using diverse frameworks. A hybrid NF implementation framework combines the available resources in the user space and the fast processing speed of the network data plane to reduce end-to-end latency and improve packet throughput and CPU utilisation.

Efforts such as HYPER [150] focus on creating a hybrid NFV framework that can leverage softwarised network functions and implementations on hardware devices without considering whether the network functions are deployed at the network data plane or in the user space. Marcuzzo et al. [151] proposed a framework for offloading parts of a virtual network function to a programmable data plane.

The proposed architecture can offload specified components of a network function and, at the same time, provide support in situations where NF offloading is not desirable. The management component of the framework comprises (1) an interface for service providers (the user module), which users can use to initiate or stop an offload request; (2) a module for translating offload code that has been compiled for installation on data-plane programmable devices, (3) a module that serves as the offload manager, for handling compiled offload code and communication with the NFV component; and (4) NFV and SDN modules for handling connections to the controller, topological data, and flow rule installation (SDN module), while the NFV module handles communication with the offload agents on the network functions.

Although the proposal presented by Marcuzzo et al. [151] aims to push some components of network functions down to the data plane of the network, it is not entirely a hybrid framework that composes packet processing pipelines from diverse NFV frameworks. Similarly, [145,150,152] attempted to solve the problem of data plane and user-space packet processing using network functions built from the same framework(s).

Some works consider the placement of network functions in a hybrid manner, i.e., using a combination of softwarised and hardware functions to create a service chain as some of the notable attempts in this domain, hybrid architectures that consider the deployment of network functions as softwarised and hardware functions were presented in [150,153–157]. [155] proposed and evaluated their ILP model using service chains that comprise hardware and software functions.

These works, however, did not look at the heterogeneity of the underlying VNF hosts and the diverse nature of service requests; the solution presented in this thesis differs in this regard – as the heterogeneity of VNF hosts, which cater for emerging use cases, is explored. The hybrid solution presented in this thesis places VNFs on user space hosts and leverages the speed of the data plane in distributed edge network environments. The heterogeneity of packet processing elements is acknowledged; in particular, the processing speed of the data plane and available resources on user space hosts are leveraged when placing virtual network functions in a hybrid manner. This approach differs from the above-mentioned efforts, which focus on hybrid placement in the context of software and hardware VNF hosts. Additionally, the hybrid placement solution in this thesis caters to the dynamic nature of edge network environments by ensuring that SLAs are met regarding subscribers' latency violations.

The genetic algorithms proposed by Cao *et al.* [154] aim to minimize link utilization and bandwidth consumption in hybrid NFV environments. They assume all network functions can be placed on VMs irrespective of type or service capability. Leivadeas *et al.* [157] focused on minimizing end-to-end delay and deployment cost by placing

network functions on cloud infrastructure and edge servers in a hybrid manner.

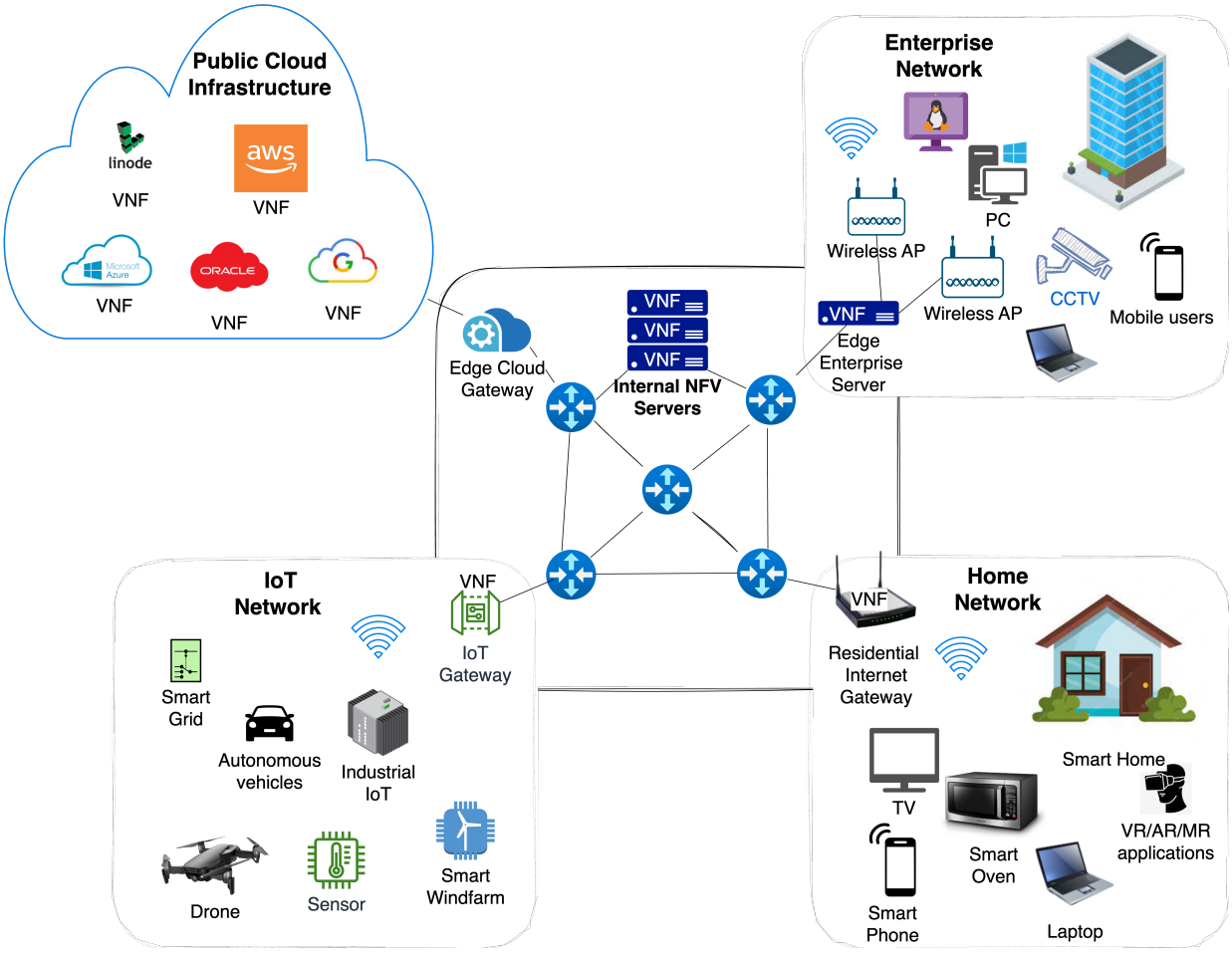

Figure 4.1: Edge Network Infrastructure

4.2.2 System Operational Overview

Operationally, the system is designed for network operators to leverage for deploying vNFs at edge network environments using heterogeneous packet processing elements. Network operators can quickly build, implement, and manage diverse vNFs on demand for heterogeneous packet processing elements.

The high number of emerging edge computing use cases makes it challenging to efficiently accommodate user vNF placement requests by network operators using the traditional edge vNF placement approach, i.e., an approach that typically lacks the proper integration of heterogeneous packet processing elements such as programmable data plane switches or SmartNICs, which should extend data plane programmability to the edge network infrastructure.

As depicted in the target network environment (i.e., the edge infrastructure in Figure

4.1), the infrastructure consists of several use case networks, e.g., a home network with everyday appliances, including smart home devices, an IoT network with devices such as autonomous vehicles and smart grid appliances. The edge network infrastructure also provides low-latency services to the enterprise network (Figure 4.1).

This network infrastructure is considered in this thesis, as it encompasses the components of a typical edge network scenario, showcasing diverse emerging use cases that can be accommodated by deploying vNFs using heterogeneous packet processing elements. The topology also captures the integration of public cloud infrastructure, which has abundant, scalable resources (compute, storage and I/O) that can be used for hosting network functions as a fall-back mechanism when edge hosts run out of capacity. Also, looking closely at the network infrastructure, operators can easily choose the composition of the internal NFV servers, including a diverse set of packet processing elements such as high-end commodity servers, programmable switch hardware, FPGA, DPDK, etc.

One goal of the proposed design is to minimise network operators' need for constant interaction to detect new applications or devices. Due to the mobility of edge users, the system needs to capture this behaviour automatically, with as minimal intervention as possible by network operators. Network operators can use code to define tailored services; thus, the platform would automatically determine the latency-optimal path while adhering to operator-defined policies, having available packet processing elements to host the defined vNF.

Since the system is designed to incorporate diverse packet processing elements, it becomes imperative to consider the complexities involved, i.e., deployability, security, and performance – for example, employing proper isolation mechanisms in the case of containerised functions (explained in §3.2.1) to ensure the security of vNFs and considering the complexities introduced while using kernel-bypass frameworks such as the Intel DPDK acceleration framework or the peculiarities of data plane programmable devices. Network operators must make required device capabilities and service adjustments to accommodate the platform based on the aforementioned operational considerations for the system to be integrated appropriately into the existing edge network infrastructure.

4.2.3 High-Level Design Requirements

An extensive review of relevant literature has been conducted to align the system's architectural design with its intended purpose, coupled with an in-depth analysis of existing vNF performance evaluation and placement platforms. The culmination of

these investigations has yielded a set of design requirements. The design requirements for the performance evaluation of heterogeneous vNF frameworks and latency-aware hybrid edge vNF placement platforms are presented below:

1. The vNF placement design should leverage heterogeneous devices for running vNFs, e.g., lightweight devices at the distributed edge network environment or the subscribers' residential network environment (depicted in Figure 4.1).
2. The support for public cloud infrastructures should be integrated as part of the platform, which should be used to place user space functions in case the network data plane runs out of capacity to host vNFs.
3. The platform should ensure the optimal placement of vNFs while considering the temporal network properties and the geographic location of subscribers. The platform should provide flexibility in the placement, prioritizing low latency paths with data plane processing elements to deliver efficient, fast services.
4. The platform should ensure subscribers stay connected to their respective vNFs at any given time by implementing transparent traffic routing.
5. The hybrid placement platform should prioritise using the network data plane to place delay-sensitive vNFs.

4.2.4 Provision for Resilience, Elasticity, and Scalability

Several factors can lead to service disruption at the network edge; some common causes of failure are the lack of resources (e.g., memory, CPU, storage, etc.) on one or more heterogeneous vNF hosts along a processing pipeline, software challenges associated with the vNFs and hardware failure. This is significantly more so because vNFs are hosted on physical devices with diverse characteristics, making edge placement solutions' resilience a key priority [158].

The elasticity of such a system is also an important consideration. The system should be able to easily adjust to changes in workload by provisioning or de-provisioning resources, based on demand, to cater to any system failures, e.g., resulting from the lack of available resources on servers to host vNFs. For example, functions requiring quick start and stop times can be implemented as containerised vNFs in the processing pipeline to achieve elasticity. Ensuring that inactive vNFs are deactivated will help achieve the system's elasticity.

Because the platform employs heterogeneous underlying hardware for hosting vNFs, designing portable vNFs by network operators that can easily be deployed on various

devices is imperative, e.g., programmable switches, containers, cloud servers, SmartNICS, on-prem VMs, etc. This will help ensure the system can easily scale by placing functions across multiple devices with unique characteristics along a given path, which adopts an end-to-end in-network compute continuum.

4.2.5 VNF Performance Considerations

Edge use cases often require the chaining of vNFs to deliver tailored services, making communication between vNFs a key design consideration for performance [83]. For example, I/O-bound functions (introduced in §3.2) require high-speed communication between an external data store and vNFs, which makes it imperative to consider, at the design stage, the communication mechanism employed to improve performance.

Furthermore, in terms of management and orchestration of functions [159], vNFs need to communicate with the NF Manager constantly [16,18,92], which often involves the creation of a shared memory space for storing metadata information, a list of service chains, and flow tables. The NF manager employs controllers in most platforms, e.g., [18, 77, 160], to handle relevant tasks such as memory management, vNF life-cycle management, inter-vNF communication, and maintaining keepalive messages.

It is also essential to note that vNF runtime performance varies depending on the vNF type and the packet processing elements' capabilities for hosting the function. For example, on the one hand, network functions such as an IDS or a DPI module, which are stateful and often computationally intensive [10, 119], can achieve improved runtime performance by leveraging the available computational resources (memory, CPU and I/O) on commodity servers. On the other hand, the performance of commonly used lightweight, stateless applications such as a stateless ACL firewall or load balancer can be improved by incorporating data plane packet processing elements (e.g., programmable switches, SmartNICs, etc.) as part of the service chain.

Another critical consideration regarding performance is always ensuring, where operationally feasible, the use of optimal locations (e.g., using metrics such as latency, bandwidth, I/O, etc.) on the edge network infrastructure for the placement of vNFs, which would reduce the need for replicating or constantly taking snapshots of vNF instances to handle resilience, thus minimising the commonly associated overheads resulting from a lack of optimal function placement [34,93,94].

As demonstrated in [93], designing NFV frameworks for resilience against node and link failure in a service chain requires additional (backup) resources to be deployed for redundancy, which comes with additional overhead if not adequately planned – becoming counterproductive in achieving high performance across the service chain.

Furthermore, not optimally placing vNFs on the edge network infrastructure has a negative ripple effect on the network's resilience, i.e., to handle downtime that is caused by inefficient function placement, network operators would have to employ several high availability and fail-over schemes, which introduces unnecessary packet processing overheads in the network [161].

By implication, to improve vNF performance and cater to emerging edge use cases, network operators that offer a chain of services, for example, comprising a gateway firewall and an IDS, can split the vNFs in a hybrid manner between the network data plane and other packet processing frameworks such as eBPF [128], ClickOS [81], etc., to reduce processing and service deployment overheads. In summary, there's a need to consider as part of the design, among other factors, 1) the runtime performance of vNFs, 2) the management of the vNF life-cycle, and 3) the high-speed communications needed between vNFs as it relates to management and service orchestration.

4.2.6 Considerations for Large-scale Deployments

Determining the physical location of functions in a given network topology is the primary goal of the ETSI NFV architectural framework (presented in §2.2.4) regarding management and orchestration. This makes it imperative that the management software handles continuity, performance management, communications between functions, and resource management. Managing large-scale deployments involving heterogeneous packet processing elements is not trivial, which is even more so as the size of the edge infrastructure grows. Referring to the target network infrastructure introduced in Figure 4.1, note that the hybrid placement scheme presented in this chapter effectively caters to large-scale deployments by employing a heuristic algorithm that handles incremental vNF placements – depending on the size of the infrastructure.

Proactive vNF management

In line with one of the promises of NFV, i.e., the ability to seamlessly adapt to network-wide changes, it is essential that the placement platform can ascertain the availability of resources on edge hosts – the placement model neatly captures this and heuristic presented in §4.3.3 and §4.4 respectively. The dynamic nature of edge users makes this consideration imperative, e.g., due to the mobility of users or traffic rerouting due to link failure, etc. Achieving the aforementioned can be complex in large-scale edge deployments, even more so with the heterogeneous packet processing elements proposed in this thesis.

Resource and VNF Placement Visualisation

Another consideration for large-scale deployments is that the network operator should

visually represent the network topology and the vNFs deployed based on user requests. This makes it easier for operators to integrate other essential components, such as a billing system or software for collecting user information. Although achieving this in large-scale deployments is not trivial, the placement model presented in this chapter allows network operators to use low-level data visualisation tools like the Python Matplotlib library to visualize users and their connection to respective vNFs.

Furthermore, in large-scale NFV deployments, a single placement visualisation interface would allow operators to manage connection states, the location of vNFs, users, and the status of services. This module would also help network operators interpret any alerts the system receives, e.g., when a user is no longer connected to a certain vNF, etc.

4.3 Latency-aware Hybrid VNF Placement

This section presents the design of the hybrid vNF edge placement on the heterogeneous host framework. The hybrid placement model is formulated as an Integer Linear Programming (ILP) problem to calculate the latency-optimal allocation of network functions on heterogeneous packet processing elements at the network edge.

4.3.1 Rationale

Due to the diverse nature of emerging applications, using the same types of underlying packet processing elements along a packet processing pipeline does not efficiently cater for use cases such as e-healthcare, autonomous vehicles, mixed reality (MR), etc. Different frameworks come with characteristics that network operators can leverage to achieve a hybrid placement of vNFs, e.g., kernel-bypass frameworks such as DPDK, which offers high performance (see Chapter 3 and §4.2.1).

In a similar context, XDP bypasses the entire network stack processing, speeding up performance. KVM machines and lightweight containers like Docker are also good candidates for hosting virtual function binaries (presented in Chapter 3). Network operators can also use software middlebox platforms such as ClickOS [81] to achieve high-performance vNF deployments. Operators can also leverage programmable devices such as P4-enabled hardware switches and FPGA-based SmartNICs to implement network functions on the other end of the spectrum of packet processing elements.

A hybrid solution can leverage the advances from the data plane and user space. To achieve end-to-end programmability at the network edge and cater for diverse use

cases, we must design placement frameworks to leverage data plane heterogeneity along a given network path. Thus, frameworks that allow us to choose the right packet processing elements are desirable based on their suitability to host a particular vNF.

As a departure from how vNFs are currently being placed, i.e., the lack of frameworks to support the seamless placement of vNFs implemented on diverse platforms, we propose a placement scheme that provisions network functions in the programmable data plane and on user space hosts to reduce packet processing overhead and overall path latency, by leveraging the benefits offered by heterogeneous packet processing elements.

4.3.2 Hybrid Placement System Model

The hybrid placement model has been formulated using the system parameters presented in Table 4.1. The graph $\mathbb{G}$ represents the physical network infrastructure's topology, having user space, data plane hosts, and edge users. The physical links between edge nodes are contained in the set $\mathbb{E}$, while paths in the network topology are captured using $\mathbb{P}$.

The model outputs a mapping of vNF requests onto heterogeneous packet processing elements and the best route (in terms of lowest delay) between source and destination nodes in the hybrid topology while adhering to the user and operator-defined constraints.

The capacity of data plane nodes (e.g., I/O, memory, SmartNIC CPU, etc.) is represented with the parameter Q_j and vNF $n_i^o \in \mathbb{N}$ bandwidth requirements by users, with the parameter b_k^{ij} (deduced from the placement requests and the network topology). The latency parameter L_k^{ij} is materialized by summing all the vNF latency requirements – depending on the application class, and the latency L_m of the network links $e_m \in \mathbb{E}$ along any given placement path. The π_i is used to handle the value of the acceptable latency threshold for each vNF, a value set by network operators based on an agreed service level agreement (SLA).

The vNF hosts requirements are captured using the parameters V_j and R_i for data plane and user space packet processing elements, respectively, and use the binary decision variable X_{ij}^k, which has a value of 1 if a vNF $n_i^o \in \mathbb{N}$ is placed on a data plane host $d_j \in \mathbb{D}$, and a value of 0 otherwise. We also introduced a second binary decision variable Y_{ij}^k which has a value of 1 if a vNF $n_i^o \in \mathbb{N}$ is placed on a user-space host $h_j \in \mathbb{H}$, and uses path $p_k \in \mathbb{P}$.

Table 4.1: System model parameters

Network entities	Description
$\mathbb{G} = (\mathbb{H}, \mathbb{E}, \mathbb{U})$	Physical network graph.
$\mathbb{H} = \{h_1, h_2, h_i, ..., h_H\}$	A set of user space hosts.
$\mathbb{E} = \{e_1, e_2, e_m, ..., e_E\}$	The physical network links.
$\mathbb{U} = \{u_1, u_2, u_o, ..., u_U\}$	User traffic associated with processing NFs.
$\mathbb{P} = \{p_1, p_2, p_k, ..., p_P\}$	Network paths from source to destination.
$\mathbb{D} = \{d_1, d_2, d_j, ..., d_D\}$	data plane elements (e.g., switches, SmartNICs)
M_k	The last heterogeneous host in path $p_k \in \mathbb{P}$
Q_j	Capacity of data plane hosts (e.g., I/O, memory, CPU, etc.) $d_j \in \mathbb{D}$.
W_i	Supported capacity of user space hosts (e.g., I/O, memory, CPU, etc.) from $h_j \in \mathbb{H}$.
C_m	Link capacity $e_m \in \mathbb{E}$.
L_m	Link latency of $e_m \in \mathbb{E}$.
VNF entities	**Description**
$\mathbb{N} = \{n_1^1, n_2^2, n_i^o, ..., n_N^U\}$	Network functions to be placed $n_i^o \in \mathbb{N}$ linked to user $u_o \in \mathbb{U}$.
π_i	Acceptable latency threshold for vNF n_i^o, based on SLA.
V_j	data plane vNF host requirements (storage, memory, CPU) of $n_i^o \in \mathbb{N}$.
R_i	user-space vNF host requirements (I/O, memory, CPU.) of vNF $n_i^o \in \mathbb{N}$.
Derived entities	**Description**
L_k^{ij}	Latency to vNF n_i^o, on a host using path p_k.
b_k^{ij}	Required user bandwidth to vNF n_i^o, using path $p_k \in \mathbb{P}$.
Decision variables	**Description**
X_{ij}^k	To denote if a particular vNF n_i^o is hosted on a data plane processing element $d_j \in \mathbb{D}$, using path $p_k \in \mathbb{P}$
Y_{ij}^k	To denote if a particular vNF n_i^o is hosted on a user space $h_j \in \mathbb{H}$ host using path $p_k \in \mathbb{P}$

$$X_{ij}^k = \begin{cases} 1 \text{ if a vNF } n_i^o \text{ is on data plane } d_j, \text{ path } p_k \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

$$Y_{ij}^k = \begin{cases} 1 \text{ if a vNF } n_i^o \text{ is on user-space } h_j, \text{ path } p_k \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

4.3.3 Placement Problem Formulation

The hybrid placement model is defined using the problem:

Consider a given network topology represented by the graph $\mathbb{G}$, with a set of users $\mathbb{U}$, a set of heterogeneous packet processing elements, and the set of individual vNFs $\mathbb{N}$, and a latency matrix l, the hybrid placement model should find the mapping of all vNFs to heterogeneous hosts that minimizes the total expected end-to-end latency from all users to their respective network functions. Equation (3) minimizes the expected user latency when placing network functions on heterogeneous hosts.

$$\min . \sum_{p_k \in \mathbb{P}} \sum_{d_j \in \mathbb{D}} \sum_{h_j \in \mathbb{H}} \sum_{n_i^o \in \mathbb{N}} (X_{ij}^k L_k^{ij} + Y_{ij}^k L_k^{ij}) \tag{3}$$

The above objective function is subject to the constraints presented next.

Capacity constraint

The limitations of the programmable data plane in terms of capacity to host vNFs must be adhered to when placing a vNF $n_i^o \in \mathbb{N}$, on packet processing elements $d_j \in \mathbb{D}$. This constraint is enforced using equation (4a) and ensuring that the resource requirements V_i of all vNFs placed are below Q_i, which represents the capability of hosts at the time of placement. In §4.3.3, a similar (but relaxed) constraint is enforced for user space nodes with vNF host requirements R_j and user space capacity W_i for hosts $h_j \in \mathbb{H}$.

$$\sum_{p_k \in \mathbb{P}} \sum_{n_i^o \in \mathbb{N}} X_{ij}^k (V_i) < Q_j, \forall d_j \in \mathbb{D} \tag{4a}$$

$$\sum_{p_k \in \mathbb{P}} \sum_{n_i^o \in \mathbb{N}} Y_{ij}^k (R_i) < W_i, \forall h_j \in \mathbb{H} \tag{4b}$$

Placement uniqueness constraint

Equation 5 ensures that all possible placements of vNFs $n_i^o \in \mathbb{N}$ on nodes are unique. This constraint checks that our hybrid placement model places a network function of a particular type on exactly one host, i.e., a data plane or user space packet processing element along a given path $p_k \in \mathbb{P}$

$$\sum_{d_j \in \mathbb{D}} X_{ij}^k + \sum_{h_i \in \mathbb{H}} Y_{ij}^k = 1, \forall n_i^o \in \mathbb{N}, \forall p_k \in \mathbb{P} \tag{5}$$

Link bandwidth constraint

This constraint captures the available bandwidth of the network links $e_m \in \mathbb{E}$. It ensures that we do not overload the edge network links along a given path (Equation (6).

$$\sum_{d_j \in \mathbb{D}} X_{ij}^k b_k^{ij} + \sum_{h_j \in \mathbb{H}} Y_{ij}^k b_k^{ij} < C_m, \forall e_m \in p_k, \forall p_k \in \mathbb{P} \tag{6}$$

Latency threshold

This constraint ensures that the vNF latency threshold Π_i is not exceeded at the time of placement for all vNFs along a path. Constraint (7) considers user latency to vNF $n_i^o \in \mathbb{N}$.

$$\sum_{p_k \in \mathbb{P}} (\sum_{d_j \in \mathbb{D}} X_{ij}^k L_{ijk} + \sum_{h_j \in \mathbb{H}} Y_{ij}^k L_{ijk}) < \pi_i, \ \forall n_i^o \in \mathbb{N} \tag{7}$$

Path validity constraint

The network path used for placement should be valid (i.e., the path should end with the host of the network function). This requirement is enforced using constraint (8).

$$X_{ij}^k, Y_{ij}^k = 0, n_i^o \neq M_k, \forall n_\iota^o \in \mathbb{N}, \forall p_k \in \mathbb{P}, \forall h_j \in \mathbb{H}, \forall d_j \in \mathbb{D} \tag{8}$$

The placement model assumes that the network backbone is appropriately configured to provide sufficient bandwidth to handle requests from delay-sensitive applications and other application types.

4.4 Hybrid Placement Heuristic Algorithm (HYPHA)

To find an optimal placement solution, the placement problem introduced in §4.3.3 requires a lot of computational time using the ILP solver, depending on the topology size (e.g., a small topology with 27 edge nodes takes $\sim$ 15 minutes to place about 100 network functions optimally and compute any latency violations etc.). Regarding time complexity, if we take the equation with the highest number of nested summations, i.e., equation 3, we cannot always assume that the operations in the loop will take constant

time, as the sizes of each set to be looped depend on network operator's implementation of the model. In general, we expect implementations to be non-linear when the entire complexity of the model is considered, i.e., including all the constraints presented in the (equation 4a to equation 8). This growth is exponential and not ideal for large-scale VNF placements, so exploring a heuristic placement algorithm with linear complexity becomes necessary. Given the nature of the network edge environment, the complexity of the exact solution does not make it ideal for large-scale network topologies – other factors, such as available computing resources and the optimisation solver used, also contribute to the overall time.

Furthermore, the optimisation model assumes the stability of network-wide information at the time of placement, which is often unrealistic due to the dynamic nature of edge network environments characterized by frequent changes in network conditions (e.g., path latency between edge nodes).

Unlike the optimal placement model in §4.3.2, the heuristic algorithm performs an incremental placement of vNFs, to find a near-optimal solution (i.e., with a relaxed optimality requirement) to the problem, which also gives network operators the flexibility to prioritise placements. The algorithm makes locally optimal decisions based on available resources and paths at each step. Requests are classified in a simple step after checking the resource requirement of each vNF (i.e., to decide if a given vNF in a set of placement requests is stateless or stateful).

The algorithm's design decisions are based on minimizing the end-to-end path latency while efficiently utilizing the available resource capacity, especially on data plane nodes, due to their resource-constrained characteristics. The design prioritizes using the shortest available paths between users and their requested functions. Additionally, the algorithm first attempts to place vNFs on selected paths consisting of data plane hosts with enough available resources, leading to faster service delivery. The heuristic also distinguishes between stateful and stateless vNFs. It handles them differently based on their resource requirements (e.g., ensuring that simple stateless functions go to the data plane while stateful functions are placed on user space nodes to leverage the high computational capacity).

Placement requests are broken down into individual vNFs to analyse the requirements defined by each user, such as delay and resource capacity requirements of vNFs that constitute a set of placement requests. Placement requests are classified in a simple step after checking the resource requirement of each vNF (i.e., to decide if a given vNF in a set of placement requests is stateless or stateful).

In Algorithm 1, `vNFRequests` consists of stateless and stateful function requests made by users S and S', respectively. The requirements for vNFs are initialised as

Algorithm 1 Hybrid Placement

```
1:  procedure PLACEVNFS (𝔾,vNFRequests, paths)
2:      Initialise data plane and user space capacity Q_j, W_i
3:      Initialise vNF requirements for data plane and user space V_j, R_i
4:      for vNF ∈ vNFRequests do
5:          paths.sort(key=len)                                   ▷ shortest paths
6:          placements ← [ ]
7:          while path ∈ paths do
8:              if d ∈ 𝔻 in path Λ V_j < Q_j: then
9:                  if checkvNFtype(vNF) == S then
10:                     d ∈ 𝔻 ← vNF                               ▷ data plane nodes
11:                     vNFRequests.remove(vNF)
12:                     updateResource(vNF_req, Q_j)
13:                     placements.append(vNF)
14:                 else:
15:                     if d ∈ ℍ in path Λ R_i < W_i: then
16:                         h ∈ ℍ ← vNF                           ▷ user space nodes
17:                         vNFRequests.remove(vNF)
18:                         updateResource(vNF_req, W_i)
19:                         placements.append(vNF)
20:                     end if
21:                 end if
22:             end if
23:             if vNFRequests then                               ▷ remaining requests
24:                 continue
25:             end if
26:         end while
27:         p ← getUsedPath(placements)
28:         latency ←computeUsedPathLatency(p)
29:         return placements, latency
30:     end for
31: end procedure
```

values based on the system's representative class of supported functions, i.e., functions in a placement request are checked against the hosts' supported and available data plane or user space capacities. Additionally, in the case of latency threshold violation, operators can set a latency bound in the form of an acceptable number that cannot be exceeded, which is required to be adhered to by the users.

The prototype network infrastructure $\mathbb{G}$, is implemented using the Python NetworkX[1] Library and uses the `shortest_path()` function to compute all possible network paths for placement. Data plane processing elements (e.g., programmable switches or SmartNICs) along a given path are checked with the available resource capacity Q_j, ensuring the constraints are respected. Note that this condition is relaxed for user space

[1] https://networkx.org

hosts based on our assumption of the abundance of resources on such hosts.

The `checkvNFtype()` function is used to ascertain if a particular vNF is a stateless or stateful function before assigning it to the right packet processing element. User space hosts $h \in \mathbb{H}$ handle the requests not fulfilled by the data plane due to the lack of available resource capacity. This process repeats until all functions in `vNFRequests` are successfully assigned.

To cater to the rare scenario of unexpected downtime on the network topology, i.e., a scenario where placement requests are not fulfilled by the data plane and user space hosts at the first instance due to hosts being unavailable, line 23 checks if this rare situation occurs and retries the placement of any unfulfilled requests until all requests are placed on either data plane or user space hosts. The most up-to-date placements are returned, with the overall latency of the used path. Depending on the service provider's policies, the placement scheme can easily be extended to accommodate specific needs (e.g., prioritising paths with the lowest latency for premium subscribed users).

4.4.1 Heuristic Computational Complexity

To determine the computational complexity of the heuristic algorithm, we analyse each step from top to bottom. Starting with the initialization steps (Lines 2-3). This involves initializing data plane capacities and vNF requirements. These steps are independent of the input size; thus, they are constant time operations, $O(1)$. The next operation sorts paths (in Line 5); this involves comparing and rearranging elements in the provided list. This operation takes $O(p * log(p))$ time, where p is the number of paths. Note that in the worst-case scenario, p can be considered proportional to the size of the network graph $\mathbb{G}$.

In line 8, checking available capacity in the data plane nodes involves inspecting each node along the network path considered. This operation is $O(n)$, where n is the number of nodes in the path. Removing processed requests (Lines 11, 16), updating remaining resources (Lines 12 and 17) and appending to current placements (Lines 13, 18) have the complexities $O(n), O(1)$ and $O(1)$ respectively, where n is the number of nodes traversed. Lines 21 to 27, verifying the remaining placement requests, getting the used paths, computing the path latency and returning the placements are all $O(1)$ operations that take linear time. The dominant factor is the iteration on $vNFRequests$ and, within that iteration, the path sorting process. Thus, the time complexity is approximately O(n * (p * log(p))), where n is the number of vNF requests, and p is the number of paths in the input graph $\mathbb{G}$.

4.5 Dynamic Hybrid Placement Scheduling

Some factors can affect the overall end-to-end latency between users and their network functions, e.g., the oversubscription of network functions placed on heterogeneous packet processing elements or the mobility of edge users due to location changes, etc. Other factors, such as the congestion of links along a service chain, can also affect a once-optimal placement.

Another key factor that could affect a once-optimal placement is the underlying structure and capabilities of the packet processing elements that make up a service chain, i.e., having a structured list of packet processing elements from different sets, supporting different functionalities, – a heterogeneous set of elements. In addition to user mobility, as the types of requests from users grow/change with time, user requests might need to be reassigned to available underlying processing elements.

Recall that in Equation 5 (capacity constraint) in §4.3.3 and the heuristic presented in §4.4, we check the capacity of the heterogeneous packet processing elements to ensure that they are not overloaded, at the time of placement, for example. By implication, having more or fewer users requesting certain vNFs over time would also result in changes to the initial hybrid optimal vNF placement.

4.5.1 Rationale for dynamic hybrid placement scheduling

The goal of the hybrid placement ILP model presented in §4.3.3 and the heuristic in §4.4 is to minimize the objective function $\mathcal{F}(L_{ijk})$, with the values of the defined binary decision variables either set to 0 or 1 at any given time. This initial optimal placement can be denoted with $\mathcal{P}_0$. We must consider a dynamic edge scenario that alters the optimal latency value l_{ijk} of users and their respective vNFs placed on heterogeneous packet processing elements using path k.

The implication of this variation in latency is that $\mathcal{F}_t = \sum_{i,j,k} x_{ijk} y_{ijk} l^t_{ijk}$ varies with time, since the initial placement $\mathcal{P}_0$, latency l^t_{ijk} varies with time t. The variation in time means that the initial placement $\mathcal{P}_0$ is likely not to minimise the new objective function $\mathcal{F}_t$, at time t. The changes in latency values also imply that the initial hybrid minimisation equation presented in §4.3.3 needs to be re-evaluated with updated latency values, which would result in a new hybrid placement $\mathcal{P}_t$ – that captures the dynamic change in latency over time.

Following the re-evaluation of the initial hybrid placement $\mathcal{P}_0$ (due to the factors explained in the previous section) and a new placement $\mathcal{P}_t$ at time t, there might be a

need to migrate vNFs between heterogeneous hosts, if we are to minimize the new objective function $\mathcal{F}_t$. For example, requests initially arriving for data plane processing elements must be migrated to available user space functions or an acceleration framework due to lack of capacity on data plane nodes or lack of support for the type of processing requested — this migration comes at a cost.

Let's consider different placements $\mathcal{P}_t$ and $\mathcal{P}_\zeta$ at different time instances t and ζ, where $\zeta > t$. The migration cost can be denoted by $\mathcal{C}_{t\to\zeta}$ and represented using the equation below:

$$\mathcal{C}_{t\to\zeta} = \sum_{ijk} P(x^t_{ijk}, x^\zeta_{ijk}) \tag{9}$$

The migration cost must be as minimal as possible to decide when to evaluate a new hybrid optimal placement $\mathcal{P}_t$. Also, we must be meticulous in determining when to perform new placement to avoid incurring additional overhead, e.g., having multiple vNFs perform redundant operations. The cost of vNF migration should be as minimal as possible, and less latency deviation should be tolerated; the expected cost of vNF migration is expressed as $\mathbb{E}[\mathcal{C}_t \to \zeta]$ = nhP$(x^t_{ijk} = x^\zeta_{ijk})$, with h as the number of heterogeneous hosts available at the time of migration, and n is the number of edge users.

One practical assumption that has been made is that a network function can only tolerate a certain level of latency change, which should not exceed its total acceptable latency threshold π_i, similar to the constraint introduced in Equation (7). A typical real-life example application could have a maximum latency allowance of 5 ms. Note that the latency values (later in Table 5.1) are of representative applications, as reported in [99, 101, 162–164], and [165].

The binary decision variable V^t_{ijk} depicts the violation tolerance at time t, which takes a value of 1 if $l^t_{ijk} > \pi_i$ or a value of 0, otherwise.

$$T^i_t = \sum_j \sum_k V^t_{ijk}, \tag{11}$$

$$L_t = \sum_i T^i_t \tag{12}$$

The tolerance for a single packet processing element is defined in Equation 11. Note that it is assumed that a function is implemented on a single host along a service chain, using a single path at any given time; thus, Equation 12 defines the latency violation tolerance for all vNFs.

It becomes essential to dynamically monitor the changes in the number of latency violations L_t defined in Equation 12 above to decide when best to re-compute the hybrid placement while keeping the cost of function migration between heterogeneous hosts to the barest minimum, i.e., acceptable SLA that would result to as minimum downtime as possible.

4.5.2 Dynamic Hybrid Placement

This section describes the dynamic hybrid placement scheduling ideal for edge network environments.

The total number of latency violations for each user can be accumulated, starting from the first optimal placement retrieved at time $t = 0$ to the current time; we represent this sum with Equation 13 below:

$$Z_t = \sum_{k=0}^{t} L_k. \tag{13}$$

From the initial hybrid optimal placement P_0 obtained from §4.3.3, an expected cost of vNF migration $\mathbb{E}[\mathcal{C}_0]$, with latency tolerance of π_i which is time-invariant, we need to ensure that the tolerance threshold is not exceeded by users sending vNF placement requests to the system. For a given time instance t of placement, if the overall latency violations (Equation 12) do not exceed the tolerance threshold, there will be no need to re-evaluate the initial hybrid placement, i.e., if $Z_t \leq \Pi$. By implication, we can avoid incurring any cost of migration $[\mathcal{C}_{0\rightarrow t}]$ if the condition mentioned above is met.

On the other hand, if the overall latency violation Z_t is greater than the tolerance threshold, we are going to incur an expected migration cost $\mathbb{E}[\mathcal{C}_{0\rightarrow t}]$, by re-computing a new optimal hybrid placement. The rate of violations can be adjusted based on particular use cases; for example, service orchestration applications would have a value of $\sim$ 20 ms, set by network operators to meet defined SLA for subscribed users.

Allowing network operators to set this threshold flexibly would also help meet the demands of diverse emerging edge use cases in future networks – supporting a hybrid implementation framework that exploits the capability of various packet processing elements.

4.5.3 Dynamic Optimal Placement Problem

The optimal time instance t^* needs to be determined for performing the optimal hybrid placement of functions while keeping the total violations Z_t close to the system toler-

ance Π as operationally possible. To avoid incurring expected migration cost $\mathbb{E}[\mathcal{C}_0]$, Z_t should not exceed the defined threshold at the time of vNF migration. To minimise the total sum of violations before getting to the maximum tolerance Π, the reward function in Equation 14 is defined, where we check the relevance of the vNF migration using $\beta \in [0, 1]$ w.r.t. the reward function. The problem becomes finding the optimal stopping time of observing Z_t that can maximise the reward function expressed below:

$$f(Z_t) = \begin{cases} Z_t & \text{if } Z_t \leq \Pi, \\ \beta\mathbb{E}[\mathcal{C}_0 \rightarrow t] & \text{if } Z_t > \Pi, \end{cases} \tag{14}$$

.

Following the abovementioned problem, we must compute the optimal stopping time t^* to obtain the smallest upper bound presented in Equation 15. This is categorised as an optimal stopping problem that seeks to stop observing the sum of latency violations presented in Equation 11 when the optimal stopping rule is found. This will trigger a new recomputation of the placement optimisation to obtain a new hybrid placement objective function.

$$\sup_{t \geq 0} \mathbb{E}[f(Z_t)]. \tag{15}$$

To avoid incurring higher vNF migration costs due to frequent migrations, we could postpone re-evaluating the hybrid optimal placement, which might lead to a higher deviation from the initial optimal hybrid placement but result in lower overhead. This is ideal to cater to the edge network's dynamic nature efficiently. The objective is to tolerate as many latency violations as possible without exceeding a predefined threshold Π.

The ideal stopping rule we aim to identify will offer an optimal approach to dynamic decision-making, maximising the expected reward, compared to any other decision-making rule. Note that Z_t is a random variable. Let's outline some fundamental concepts from the theory of optimal stopping before delving into proving the uniqueness and optimality of the proposed rule.

4.5.4 Optimal Stopping Theory Concepts

Optimal Stopping Theory involves continuously tracking a random variable to decide whether to perform a specified action. The goal is to reduce the expected cost or to increase the expected payoff for making a certain decision. [166]. The theory is ideal for scenarios involving different stages in the decision-making process, i.e., where the

result of the next stage can be estimated but not fully predictable. In general terms, the goal is to define an optimal rule in which a decision can be taken to either reduce cost or increase a defined reward function.

To define an optimal stopping problem, two main components are used: (1) a sequence of random variables Z_1, Z_2, assuming that the joined distribution of such variables is known, and (2) a sequence of reward functions values, depending on the observed random variables. The observed random variable is either minimised or maximised as an objective.

We observe the sequence of random variable $(Z_t)_{1 \leq t}$, in our case, and we can decide to stop at any given instance of time t and implement our decision, i.e. the re-computation of the hybrid optimal placement and a possibility of migrating vNFs between heterogeneous hosts. A reward function $f_t \equiv f(z_t)$ is induced if we decide to stop observing the random variable at time t. Some definitions are presented below.

4.5.5 Definitions and theorems

Let's find the optimal stopping time t^*, which maximises the expected reward function. With the value of ζ possible to be ∞: $\mathbb{E}[f_t^*] = sup_0 \leq t \leq \zeta \mathbb{E}[f_t]]$.

A sequence $\mathbb{F}_t$ of values from the random variables $Z_1, ..., Z_t$.

The stopping criterion is referred to by the 1-stage look-ahead stopping rule in Equation 17.

$$t^* = \inf\{t \geq 0 : \ f_t \geq \mathbb{E}[f_{t+1}|\mathbb{F}_t]\} \tag{17}$$

t^* calls for stopping at a given time instance t where the reward for stopping is as high as the expected reward for moving to the next instance $t + 1$.

An event can be denoted with E_t, $\{f_t \geq \mathbb{E}[f_{t+1} \mid \mathbb{F}_t]\}$. We express the stopping rule problem as monotone if the condition $E_0 \subset E_{t+1} \subset E_2 \subset ...$ is almost surely (a.s). This problem is expressed as E_t, known as the given set in which the one-step lookahead (1-sla) rule stops at time t. The condition $E_t \subset E_{t+1}$ implies that if the one-step lookahead rule results in a stop at time instance t, it will stop at $t + 1$ regardless of the value of Z_{t+1}. In a similar context, for $E_t \subset E_{t+1} \subset E_{t+2} \subset ...$, it implies that a call for a stop by the 1-sla rule at time instance t means it will also call for stopping at all future instances, regardless of the values.

A 1-sla rule is presented, which is optimal for the problem in 4.5.3. Note that the I-sla rule is optimal for monotone-stopping problems [167]. □

4.5.6 Hybrid VNF Placement – Optimal Scheduling

The hybrid optimal placement of vNFs is re-computed at the optimal stopping time t^*. This implies that a new hybrid optimal placement P_{t^*} will be derived, with $\lambda > 0$; thus incurring an expected cost of vNF migration $\mathbb{E}[\mathcal{C}_0]$.

The hybrid optimal placement $\mathcal{P}_0$ at $t = 0$ is re-computed at time t, thus:

$$\inf_{\mathcal{T} \leq 0} \{\zeta : \sum_{\ell=0}^{\Pi - Z_{\mathcal{T}}} \ell P(L = \ell) \leq (Z_\zeta - \lambda \mathbb{E}[\mathcal{C}_0])(1 - F_L(\Pi - Z_\zeta))\} \tag{18}$$

where $F_L(\ell) = \sum_{l=0}^{\ell} P(L = l)$ is the cumulative distribution and $P(L = \ell)$ is the mass function of L in Equation 12.

Proof. Theorem 2: The aim is to find a condition in Equation 17 to stop monitoring the variable Z_t, and re-compute the vNF placement. The target becomes the expected value $\mathbb{E}[f(Z_{t+1}) \mid Z_t \leq \Pi]$, from $\mathbb{F}_t$, up to time instance t, thus:

We stop at time t, where $\mathbb{E}[f(Z_{t+1}) \mid Z_t \leq \Pi] \leq Z_t$, i.e., to derive the one-step lookahead:

$$\sum_{\ell=0}^{\Pi - Z_t} \ell P(L = \ell) + (Z_t - \lambda.\mathbb{E}[\mathcal{C}_0]) F_L(\Pi - Z_t) + \lambda.\mathbb{E}[\mathcal{C}_0] \leq Z_t,$$

the interested reader is referred to [168] for proof.

□

The above theorem implies that when we re-evaluate the hybrid optimal placement at time $t = 0$, we stop when the condition in Equation 18 is met, i.e., at the first instance of time $t \geq 0$. The expected reward in Equation 14 is thus maximised, based on the aforementioned criterion.

Since the second problem presented in §4.5.3 is monotone, based on Theorem 1, the 1-sla stopping rule is optimal. We can thus stop re-evaluating the initial optimal hybrid placement at the first instance of time t, where $f + t(Z_t) \geq \mathbb{E}[f_{t+1}(Z_{t+1}) \mid \mathbb{F}_t$

i.e., with $\{Z_t \leq \Pi\} \in \mathbb{F}_t$. By implication, no further maximisation of the reward function will be obtained by continuing to re-evaluate the hybrid optimal placement.

We can prove that the hybrid vNF placement, i.e., the second problem, is optimal, using the 1-sla rule in Equation 18.

Proof. Theorem 3: For the 1-sla rule to be optimal based on Theorem 1, it implies that the difference $E[f_{t+1}(Z_{t+1}) \mid \mathbb{F}_t$ is monotonically non-increasing with $Z_t \in [0, \Pi]$, which makes the 1-sla rule optimal for the second problem presented in §4.5.3.

□

The overall number of diverse vNFs directly affects the computational complexity of the evaluation. Regarding complexity, we do not expect the stopping criterion evaluation to be complicated; it avoids any complex decision-making process when triggering the re-evaluation of the hybrid optimal placement. As shown in Equation 18, the summation from 0 to $\Pi - Z_k$ at time k determines the stopping criterion. The sum at time k can be recursively evaluated from the sum to $k - 1$. Thus, $O(N)$ is the time complexity, with N representing the number of diverse functions.

4.6 Summary

This chapter discusses how network operators can deliver faster service while catering to diverse vNF requirements in emerging edge network environments. In summary, this can be achieved by leveraging the network data plane and other packet processing acceleration frameworks, such as DPDK, eBPF, FPGA-based NICs, ClickOS, etc., for deploying virtual network functions. The chapter presented the outcome of investigations into the diverse nature of user vNF placement requests based on emerging use cases such as e-healthcare, autonomous vehicles, etc.

The chapter presented the key design requirements and the need to use heterogeneous packet processing elements to deploy virtual network functions at the network edge. The technical argument for a hybrid placement scheme on heterogeneous hosts is also presented in §4.2.1.

A hybrid framework's essential requirements and design concepts for placing vNFs on heterogeneous packet processing elements (e.g., lightweight containers, SmartNICs, programmable switches, VMs, etc.) were presented in this chapter (§4.2.3). The critical considerations for designing such a system were presented in §4.2, including the considerations for large-scale deployments (§4.2.6) and some key vNF performance considerations (§4.2.5).

A hybrid vNF placement optimisation model and heuristic that leverages the low-latency, high-speed feature of the network data plane and the abundant resources on user space hosts is proposed in §4.3.2. The hybrid placement model design is presented in §4.3, which captures the rationale and problem formulation.

The heuristic solution presented in §4.4 is designed to cater to real-world networks' dynamic nature, which makes using optimisation models challenging; thus, HYPHA can quickly obtain near-optimal vNF mapping while incurring fewer latency threshold violations set by network operators (§5.2.2 and §5.2.4).

In §4.5, using the principles of optimal stopping theory (OST), we propose an optimal placement scheduling model to handle dynamic edge placement scenarios while simultaneously minimising the number of latency violations and vNF migrations based on set SLAs by network operators.

In summary, the design principles presented in this chapter will help network operators to deploy diverse network functions on heterogeneous hosts at the distributed network edge infrastructure while catering to the dynamic nature of the distributed edge network environment. The evaluation results of the proposed model are presented in the next chapter.

Chapter 5

Edge Hybrid VNF Placement Framework Evaluation

5.1 Overview

Following the hybrid placement model and heuristic design presented in the previous chapter, this chapter describes the experiments designed to evaluate the hybrid edge placement solution. It also describes the details of the evaluation environment designed to evaluate the proposed placement scheme, including the network topology used, the link latency modelling, and the use case application's latencies considered.

To show the benefits of running vNFs on heterogeneous hosts, two unique placement scenarios are first evaluated in §5.2.1, i.e., user-space-only and hybrid vNF placements, particularly to ascertain the average latency of users to their vNFs. The performance of the heuristic algorithm and the optimisation model is compared in §5.2.2 (in terms of objective functions), using three main scenarios, i.e., data-plane-only, hybrid and user-space-only placements.

To gain insights on the percentage of vNF placement requests fulfilled, the experiment in §5.2.3 was carried out. The performance of the heuristic solution to the optimisation model is compared by considering a scenario with dynamic latency behaviour in §5.2.4, particularly in handling latency violations.

The performance of the dynamic placement model is compared by considering a scenario with dynamic latency behaviour in §5.2.5. In §5.2.6, we considered the cost of vNF migration while using two different placement scheduling approaches, i.e., scheduling placements after a fixed time interval and scheduling using optimal stopping time.

The experiments carried out in Chapter 3 using real hardware to evaluate the performance benefits of deploying network functions on DPDK, containers, switch data planes, and virtual machines informed some of the decisions for the experiments designed and presented in this chapter. Specifically, it provided useful insights into what network functions can perform better when deployed on specific packet-processing frameworks (e.g., the use of DPDK to speed up the performance of network functions in terms of the number of packets processed per second and the use of switch data plane to deploy simple stateless network functions). This also motivated exploring a hybrid deployment scheme in the simulation-based experiment to leverage characteristics of the representative frameworks evaluated using real hardware. For consistency, the experiments designed in this chapter consider other factors, such as using real latency measurements built on a network topology obtained and extended from a real nationwide backbone network infrastructure.

5.2 Evaluation Environment

The system evaluation environment is presented, including bandwidth, topology and latency values. The trace-based simulations were run on an Apple MacBook Pro with the M1 Pro Chip, a 10-core CPU and 16-core GPU, 16GB unified memory, and Gurobi optimisation solver version 10. The model and heuristic were implemented in Python, with the topology implemented as a graph modelling language (GML) file using the NetworkX Python library, which easily allows for building and manipulating network models.

For the experiments in this section, users are assumed to have VNF requests that fall under the categories in Table 5.1; VNF placement requests are simulated based on the representative latency bounds; and in some experiments e.g., in 5.2.3 where the number of fulfilled placement requests are evaluated, the generated VNF requests are either categorised as stateful (a representative category of network functions requiring a lot of computational capacity, e.g., a deep packet inspection module) or stateless (a representative category of less computationally intensive function, e.g., a simple access control function) ideal for data plane implementation.

Note that (as depicted in Figure 5.3) a Gamma distribution was used to simulate real-life scenarios where stateless VNFs have a much higher occurrence in service provider environments. The values for latency threshold are modified for experiment runs, depending on the network function simulated, e.g., a latency value of 5 ms is used when simulating the behaviour of autonomous vehicles, VNF placement requests, etc.

- **Network topology and latency bounds**

To obtain practical and useful results for network operators, a real-world network topology from topology zoo[1] was used for the evaluations. The hybrid network topology has been modelled by capturing the peculiar characteristics and constraints of the network data plane and introducing compute capacities on edge nodes, representing the network data plane. Table 5.1 shows some of the characteristics of representative vNF use cases [101, 165], with latency thresholds ranging between 5 to 80 ms.

Table 5.1: Common use case latency characteristics

Example application	Latency class	Latency bound
Autonomous vehicles	Real-time	5 ms
Smart Grids	Low (near real-time)	10 ms
Mixed Reality (AR & VR)	Low (soft real-time)	10 - 15 ms
Service Orchestration	Near real-time	20 ms
Monitoring applications	Non real-time	80 ms

The nationwide Jisc NREN UK backbone network topology was used by adjusting the bandwidth parameter and topology size to fit edge DC environments and the use case of having heterogeneous nodes. The bandwidth limit between vNF hosts is 100 Mbit/s or 1Gbit/s – a typical bandwidth range for most edge computing use cases [101, 169]. Note that this also depends on the vNF bandwidth requirements and the supported bandwidth of the actual physical topology. Compute capacity is provisioned on edge devices, and the data plane is assumed to have finite computing resources (i.e., I/O, memory, storage, etc.) with ample resources on user space hosts for hosting diverse network function types.

– **Link latencies modelling**

The values of link latency are sampled from a Gamma distribution using the parameters (k = 2.2, θ = 0.22) as in [101] and [99], based on the nature of real-world latency values obtained from PerfSonar with average Hurst value of 0.6. Thus, a time series is created by sampling from this distribution (with a value of 100 in the experiment), representing the link latency values over time. This time series reflects periods of low latency and occasional high latency spikes – a typical pattern in edge/WAN networks due to traffic rerouting, avoiding congested paths, etc. The values were modelled from real-world latencies observed from the Energy Sciences Network (ESnet) PerfSonar[2] measurements.

[1] http://www.topology-zoo.org/
[2] http://ps-dashboard.es.net/maddash-webui/

5.2.1 User-space-only and hybrid placement latency

In this evaluation, the user latencies of placing network functions in two main scenarios are shown; the first scenario is where all vNFs are deployed on a given set of user-space edge nodes, which benefits from support for network functions with diverse capacity requirements.

A second scenario is in which some network functions are deployed on data plane nodes (in a hybrid manner), characterised by having low packet processing delay, with the downside of limited resource capacity to support a diverse set of network functions. In this evaluation in the hybrid network, when data plane nodes are out of processing capacity to host the requested vNFs (i.e., due to their resource-constrained characteristics) – user space hosts are used.

The placement model presented in §4.3.2 is implemented using a commercial optimization solver, Gurobi[3]. In this experiment, edge users were assigned a typical last hop latency of 2 ms to their respective vNFs. To cater for a diverse set of network functions, vNFs were assigned to each user with different latency requirements. This is a typical scenario for vNFs with user space and hybrid requirements. Computing resources were assigned to the data plane and user space nodes in the topology; 500 vNF placement requests were generated in total for this experiment.

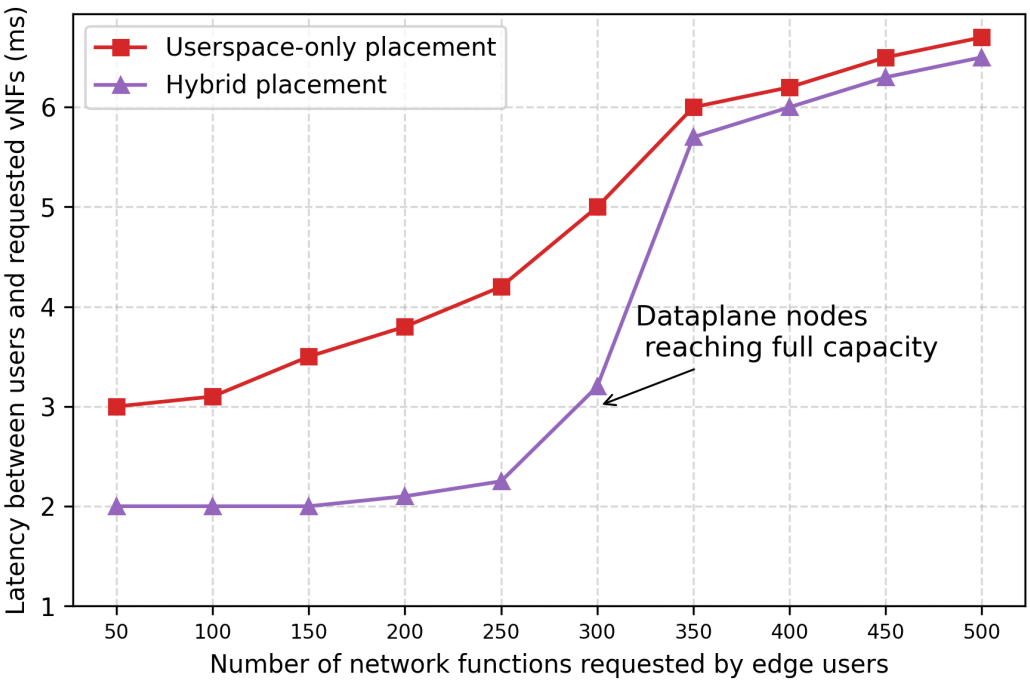

Figure 5.1: Average user to vNF latency in user-space-only and hybrid optimisation placement scenarios.

[3]https://www.gurobi.com/

As depicted in Figure 5.1, placing the requested functions using the hybrid topology yields the lowest latency, which deteriorates as the number of vNFs grows ($\sim$ 300 vNFs). This is attributed to the low resource capacity of the available data plane nodes closer to the users along a chosen network path. The user space-only placement results in much higher latency, although this comes with the advantage of fulfilling more diverse vNF placement requests (demonstrated in the evaluation carried out in §5.2.3) due to the ability of such hosts to handle all types of network functions, that are not suitable for data plane nodes. The hybrid placement scenario (purple line in Figure 5.1) presents a *sweet spot* in terms of reduced latency, as it presents a trade-off between the resource-constrained nature of the data plane nodes and the high latency of the user space nodes.

5.2.2 Optimisation and heuristic objective functions

In this evaluation, the values of the objective functions for all placement scenarios are considered for the optimisation and heuristic placements, focusing on analysing how the overall objective function (end-to-end latency) varies in each scenario. The exact number of requests is maintained as with previous experiments (500 vNFs) and focused on evaluating the effect of capacity constraints. As depicted in Figure 5.2, the resource capacity of the edge nodes in the topology is varied – specifically, network functions were assigned using data-plane-only, hybrid and user-space-only scenarios. This made it possible to assess the model's performance and heuristic solutions under different resource constraints.

In the first hybrid placement scenario, data plane nodes can host 50% of the total vNF placement requests; we reduced this number in the second hybrid placement scenario to 30% to observe how it affects the objective function and finally to 0 (in the user-space-only scenario).

The data-plane-only (normalised values of 0.17 and 0.22 for heuristic and optimisation, respectively) scenario yields the lowest end-to-end overall latency, 78% better than the user-space-only placement scenario (normalised values of 0.95 and 1 for heuristic and optimisation, respectively). The best-case scenario in hybrid placements is the first hybrid placement, i.e., the scenario where 50% of the total placement requests are pushed down to the data plane packet processing elements. This scenario, represented with normalised values of 0.43 and 0.48 for heuristic and optimisation, respectively, is 48% better than the user-space-only placement. With fewer placement requests hosted by data plane hosts, i.e., the scenario with 30% of the requests on data plane processing elements, hybrid placement scenario 2 produced a much higher objective function, i.e., normalised values of 0.64 and 0.69 for heuristic and optimisation,

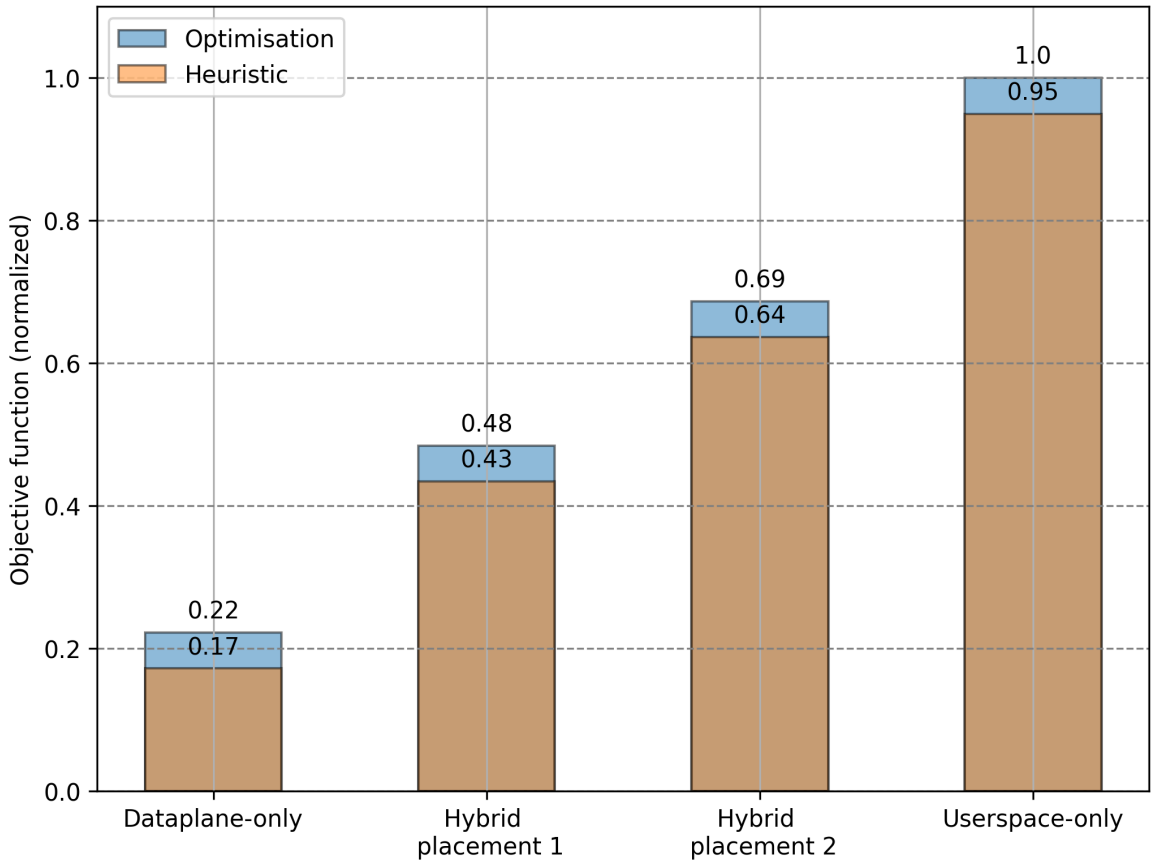

Figure 5.2: Heuristic and optimisation objective functions for different placement scenarios.

respectively.

The overall phenomenon observed in this evaluation is attributed to the preference for data plane edge nodes over user space nodes, particularly in hybrid placement scenarios where a combination of user space and data plane packet processing elements are used for placement. Note that the hybrid scenario also provides the core advantage of encompassing edge nodes with heterogeneous capabilities – more delay-sensitive applications can thus be pushed down to the network data plane to be processed quickly. Furthermore, depending on the capabilities of the host architecture, in packet processing frameworks that do not support multiple receive and transmit queues on NICs (essentially sub-Gigabit Ethernet NICs), complex processing is better executed on user space packet processing elements to leverage the availability of multiple CPU cores in modern architectures [40]. The heuristic placement yields near-optimal objective function values in all scenarios, which also yields results faster when compared to the exact solution – a topology with 27 edge nodes takes 45 seconds to compute the placement of 100 functions. In contrast, the same number of nodes and functions require $\sim$ 15 minutes using the exact solution, making the heuristic solution faster and thus ideal for large-scale deployments.

5.2.3 Number of placement requests fulfilled

This evaluation investigates the percentage of placement requests that can be fulfilled. Unlike the previous experiments, this experiment focuses on the heterogeneity of the placed vNFs (in terms of the types of vNFs) while keeping other components of our evaluation the same, i.e., topology size, temporal link latency, nodes resource capacities, etc., for consistency.

500 user vNF placement requests were generated using a Gamma distribution with k=1.6 and $\theta = 0.5$ for stateless functions and k=2.5 and $\theta = 1$ for stateful functions. The Gamma probability distribution captures the nature of commonly used network functions, where many are simple stateless functions deployable in the data plane [119]. Thus, the heavy-tailed distribution values for stateless functions in the evaluations represent this phenomenon (i.e., a high occurrence of such functions).

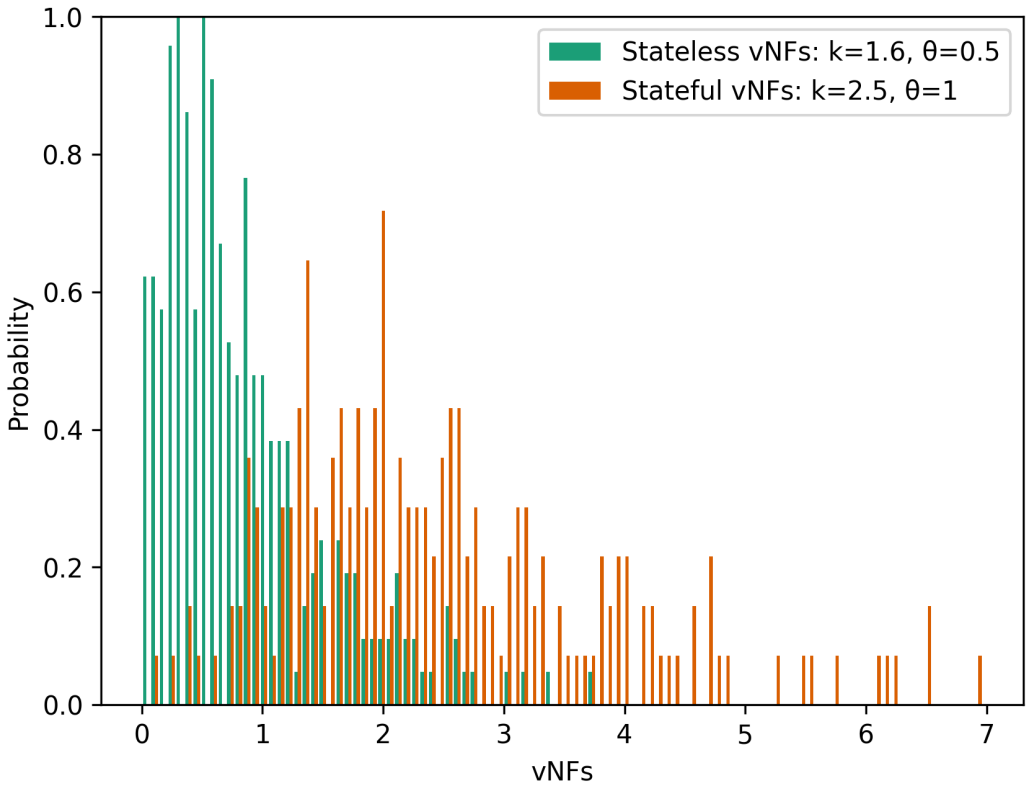

Figure 5.3: Probability distribution functions of stateless and stateful vNFs

Figure 5.3 and 5.4 depict the probability distribution functions of vNFs and the evaluation outcome, respectively. The nature of our placement requests mimics real-life vNF placement requests sent in by users (i.e., providers often do not have prior knowledge of the types of functions that are requested); also, a fair estimation of the phenomenon found in commonly used network functions [119], e.g., header classifiers, stateless

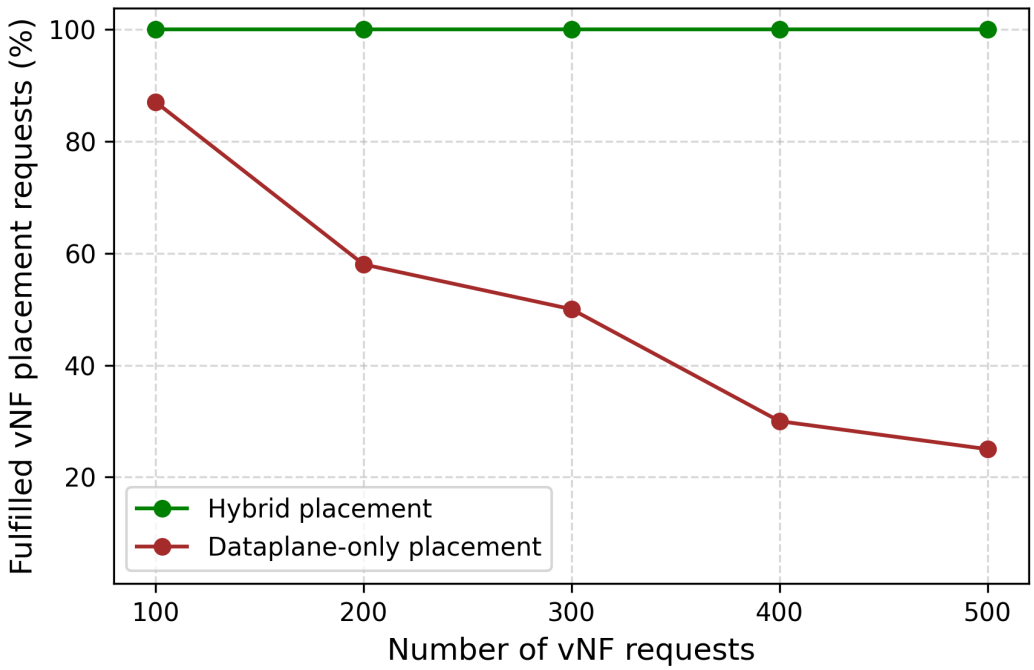

Figure 5.4: Comparison of the number of fulfilled vNF placement requests in hybrid and data plane-only scenarios, with different numbers of user placement requests.

match-action firewalls, and load balancers – these functions are often not compute-intensive.

The number of user vNF placement requests is analysed in different placement scenarios, i.e., hybrid and data plane-only placements. The resource-constrained nature of the data plane edge nodes means fewer placement requests are handled successfully by nodes such as programmable switches or SmartNICs (Figure 5.4). The hybrid placement scenario can maintain many fulfilled requests in our evaluation. This is attributed to the ability of our model to utilize both the data plane and user space nodes in the hybrid placement scenario. The hybrid placement scenario can handle a *mix-and-match* of vNFs along a given path, which results in fulfilling *all* the placement requests received. On the other hand, the constrained nature of the data plane nodes does not allow vNFs with high computational requirements (or stateful functions) to be placed; this is observed particularly as the number of vNFs requested by users grows.

As depicted in Figure 5.4, the hybrid scenario maintains many fulfilled requests, attributed to the support for *all* types of network functions. Although the number of fulfilled placement requests is lower in the data-plane-only scenario, it offers reduced latency (as demonstrated in §5.2.1). More than 60% of received requests are processed in the data plane, particularly for < 300 requests; this number drops as the number of placement requests grows, and fewer stateless network functions are found in the requests.

5.2.4 Dynamic latency violations

The dynamic nature of mobile edge users often results in changes in the temporal latency values over time, i.e., due to the mobility of users. Note that vNFs have a latency threshold value $\Pi_i > 0$ (defined in §4.3.3), which is set by network operators (e.g., based on latency SLA between providers and customers) – violating this set latency threshold could result in application performance degradation.

The number of latency violations is analysed over a time series; a time instance in the series could be, e.g., 3 minutes in real-world edge network environments. The value of π_i is set at 30 ms for this evaluation (note that network operators can define this threshold based on the agreed SLA). By incorporating a time series component, a hybrid placement scenario that expands upon evaluating the heuristic and optimal objective functions presented in §5.2.2 is employed. Precisely, the latency violations at each time instance are tracked.

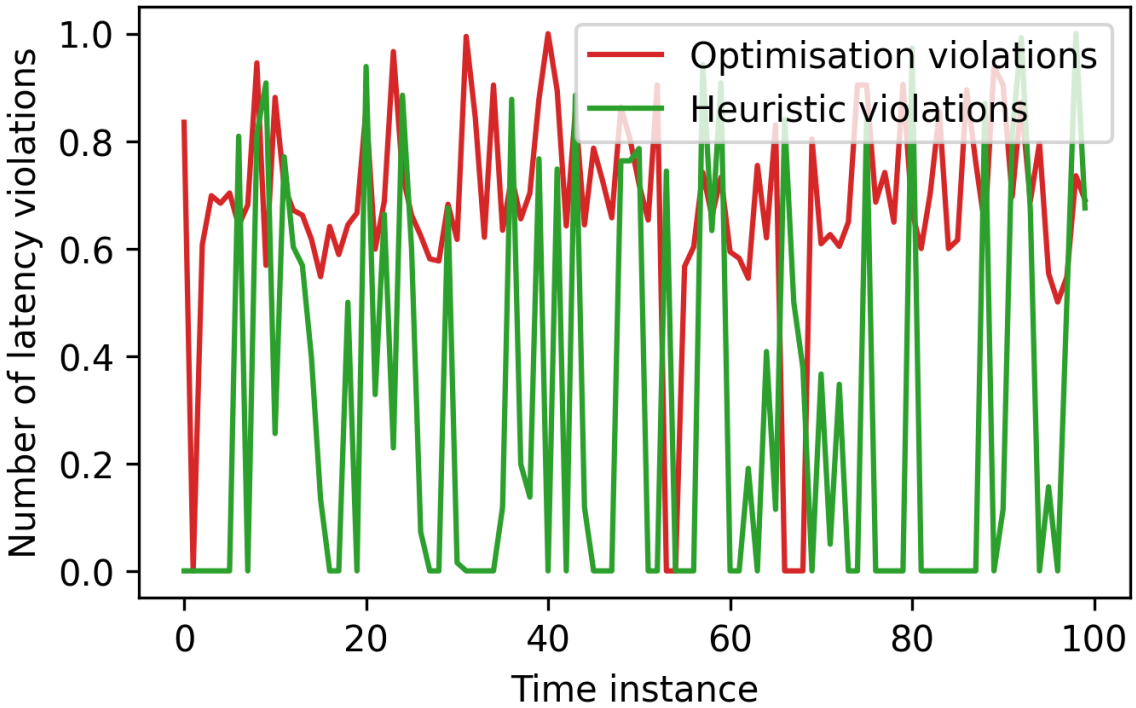

Figure 5.5: Heuristic and optimisation latency violations performance.

In Figure 5.5, the latency violations over the time series are shown to evaluate the ILP model and heuristic performance and provide service providers with valuable insights into application latency violations. Note that the hybrid placement heuristic presents fewer latency violations – this is attributed to the incremental placement mechanism employed by the heuristic, which prioritises the available shortest network paths containing data plane processing elements at the time of placement, i.e., unlike the ILP

model, which tries to compute the end-to-end latency by considering the *global* network parameters, slower to react changes in the network; thus leading to a higher number of latency violations at several time instances.

5.2.5 Hybrid Placement Scheduling With Optimal Stopping Time

The dynamic nature of mobile edge users often results in changes in the temporal latency values over time, i.e., due to the mobility of users. Note that vNFs have a latency threshold value $\pi_i > 0$ (defined in §4.3.3), which is set by network operators (e.g., based on latency SLA between providers and customers) – violating this set latency threshold could result in application performance degradation.

We analyze the number of latency violations over a time series; a time instance in the series could be, e.g., 3 minutes in a real-world edge network environment. We set the value of π_i at 40 ms for this evaluation (note that network operators can define this threshold based on the agreed SLA).

We evaluate using the proposed optimal scheduling model to run the experiments for 1000-time instances. Note that for this experiment, the topology used already comprises heterogeneous packet processing elements – from the previous evaluations; hence, our focus is on evaluating the dynamic scheduling solution, i.e., we investigate the behaviour of the system while taking note of the accumulated latency violations before reaching the system's latency tolerance threshold Π, which service providers can flexibly specify.

Unlike the previous experiment, we monitor the total number of latency violations over each time instance, ensuring that the threshold is not exceeded. Note that a suitable use case for the latency threshold is distinguishing the SLA between different classes of users, e.g., users that have paid for premium service could be given a custom-defined latency violation threshold for their applications by network operators.

Recomputation of the existing placement is triggered after some time; thus, vNF migration between heterogeneous hosts is performed to obtain a new optimal latency after migration. The core of this evaluation is also to understand the impact of unpredictable deviation in latency over time due to the factors presented in §4.5.

In figure 5.6, we depict the behaviour of our dynamic placement scheduling solution over 1000 time instances, i.e., minimising the number of placement computations while not having any latency violations. For the time instance used for the evaluation, we avoided reaching the latency threshold in three instances (where new placement is re-computed). Optimal placement re-computation is performed at least three times

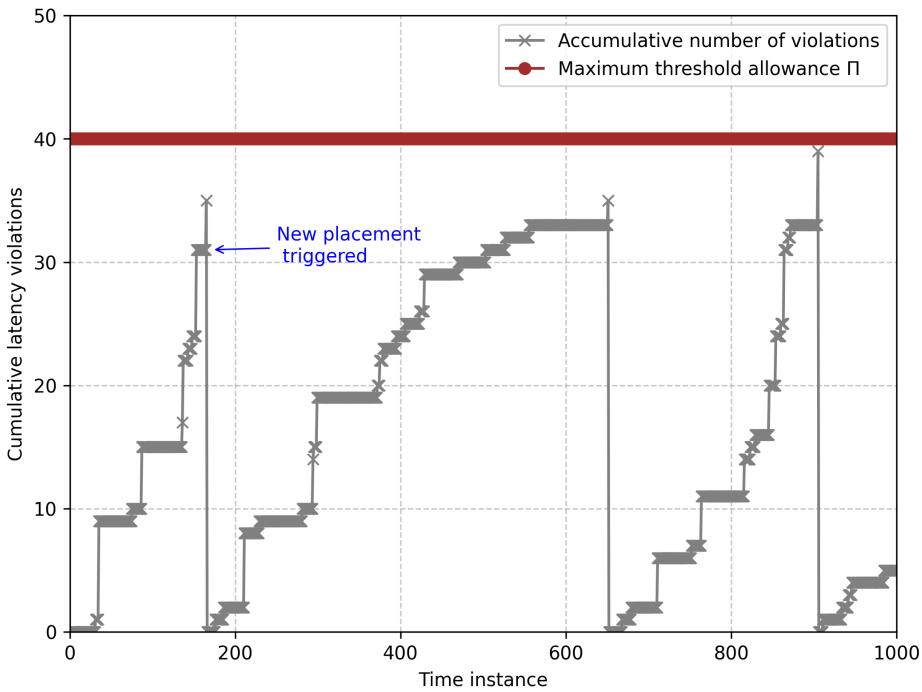

Figure 5.6: New placement scheduling is dynamically triggered before getting to the latency violation threshold.

over the time instance used in this evaluation, which avoids reaching the latency violations threshold in all cases.

5.2.6 Number of VNF Migrations

In this evaluation, we look at the number of vNF migrations between heterogeneous nodes, i.e., due to the hybrid optimal placement re-evaluation. Functions must migrate, e.g., from a programmable data plane packet processing element to a userspace host. We compare two main scenarios of vNF migrations, i.e., 1) migrating vNFs using the optimal stopping time model presented and 2) migrating vNFs over every 50-time instances.

As used in our previous experiments, a time instance can be a time of 3 minutes in real-life edge network environments, which might result in changes in latency. As shown in figure 5.7, unlike scheduling the placement at every time instance, which results in migrating more vNFs, thus incurring higher vNF migration costs, placement scheduling using optimal stopping time significantly reduces the number of vNF migrations. To achieve fewer vNF migrations, the accumulated latency violations from the previous evaluations are used, keeping the threshold below the value Π defined by service providers.

From Figure 5.7, note that operators can determine the fixed time interval for vNF

migrations between heterogeneous hosts, which can easily fit into use cases that require vNFs to only be spurn up at specific times, e.g., functions that perform overnight system backups or applying security patches at defined periods. The optimal stopping time vNF migration significantly reduces the migration cost compared to scheduling at specific time intervals (Figure 5.7).

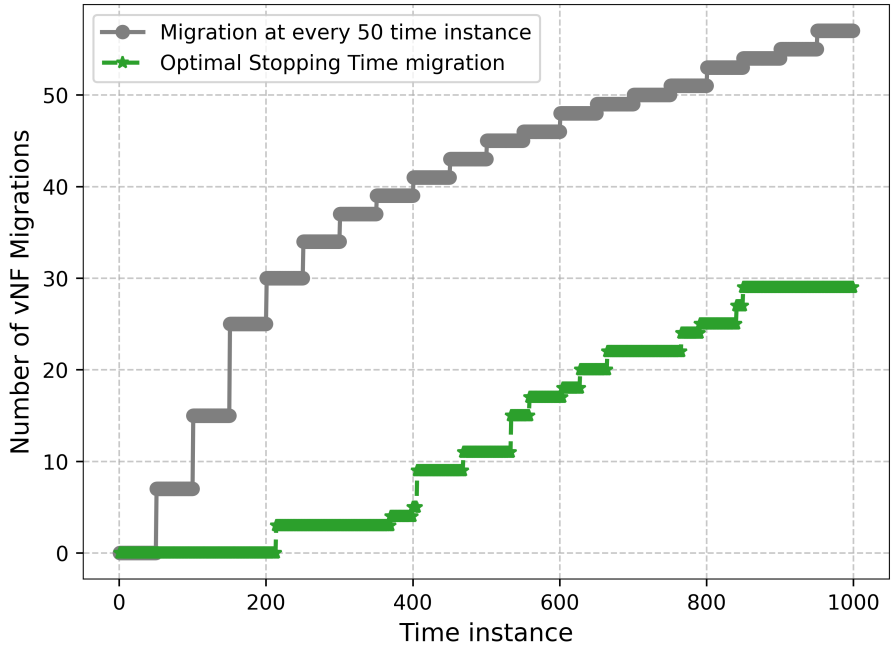

Figure 5.7: Total number of vNF migrations using Optimal Stopping Time and scheduling at every 50-time instances.

5.3 Summary

The chapter presents the environment for evaluating the dynamic latency-aware hybrid placement model and the hybrid placement heuristic algorithm. Evaluation details were also presented, such as the network topology and latency bounds of use case applications.

Through extensive evaluations presented in this chapter, it has been demonstrated that employing a hybrid deployment scheme that leverages the processing capability of diverse acceleration frameworks yields minimal user-to-vNF latency and overall end-to-end path latency, thus fulfilling the placement of a diverse set of vNF requests from emerging edge computing use cases.

Results for the evaluation presented in this chapter show that network operators can leverage the high-speed, low-latency feature of data plane packet processing elements

to host delay-sensitive applications and improve service delivery for subscribed edge users with diverse requirements.

The dynamic hybrid placement scheduler evaluated in this chapter can efficiently handle changing placement requests from edge users. The proposed scheduler reduces the number of vNF migrations between heterogeneous hosts compared to other approaches that perform vNF migration at set time intervals.

Chapter 6

Conclusion and Future Work

6.1 Overview

This thesis's final chapter outlines the research contributions in §6.2. The thesis statement is revisited in §6.3, and possible extensions to this work are identified in §6.5, which includes intelligent function placement in a hybrid network environment that leverages the heterogeneity of edge hosts. Some essential use cases for the hybrid placement framework are described in §6.4, which also discusses how network operators can benefit from the outcome of this work as they tackle the challenges in next-generation networks.

6.2 Contributions

The motivation for the work presented in this thesis has been derived from an in-depth review of existing literature in Chapter 2, which critically analysed the frameworks used by network operators to deploy virtual network functions in next-generation network environments. Chapter 2 also presented a comprehensive state-of-the-art review of the relevant literature in the areas of vNF and SFC deployments to build a strong understanding of the most recent efforts in this domain, including a proposal of a novel taxonomy of frameworks and a thorough review of the status quo in edge vNF deployments by researchers and network operators [3].

The thesis leveraged the paradigm and principles of network function virtualisation, software-defined networking, data-plane programmability, and edge computing to propose a novel optimisation model for hybrid vNF placement at the network edge. The work presented in this thesis motivates the concept of incorporating fast data plane programmable devices and other lightweight packet processing acceleration frameworks,

such as DPDK and containers, in adopting an end-to-end programmable pipeline that extends to the network edge. This thesis advocates advancing function virtualisation and deployment by exploiting network programmability over diverse emerging environments, such as programmable switch hardware and in-network acceleration platforms.

In Chapter 3, a critical analysis of the nature of virtual network functions and the heterogeneity of packet processing elements was carried out. This analysis extends to the composition and complexity of SFCs and edge placement environments, including a classification of representative virtual network functions in future networks based on emerging edge computing use cases. The design of a real testbed for evaluating the performance of heterogeneous virtual network functions using diverse packet processing elements was also presented. Results from evaluating a diverse set of network functions are also presented, i.e., performance measurements from leveraging a heterogeneous set of packet processing elements in next-generation networks [10].

Chapter 4 presents the motivation for considering the network data plane and other in-network acceleration frameworks for vNF placement and why a hybrid placement scheme is needed to meet the diverse demands of edge computing use cases in next-generation networks. The fundamentals for latency-aware hybrid placement of vNFs on heterogeneous hosts are also presented in this chapter, which addresses the lack of frameworks that can leverage diverse packet processing acceleration technologies to offer end-to-end programmability along a given network path in the edge network infrastructure.

This chapter also presents a novel formulation for the latency-aware hybrid placement of virtual network functions problem and an exact solution using Integer Linear Programming [170]. In addition to the optimization model presented, a novel hybrid heuristic placement algorithm is proposed to cater to the high computational requirements of solving optimization models and some assumptions made by such models, e.g., the stability of network-wide information necessary for the optimal placement of network functions. The hybrid placement heuristic algorithm also aims to cater to the dynamic nature of edge network environments, which often requires path recalculation due to factors such as the mobility of edge users in next-generation networks. Using the principles of optimal stopping theory (OST), an optimal placement scheduling model was presented in this chapter to handle dynamic edge placement scenarios while simultaneously minimising the number of latency violations and vNF migrations based on set SLAs by network operators.

In Chapter 5, a comprehensive evaluation was performed to ascertain the benefits in terms of the performance of the proposed dynamic latency-aware hybrid edge vNF

placement model. To find a quick and efficient solution to the hybrid vNF placement problem, the performance of the proposed placement heuristic algorithm was also presented using real service provider network conditions.

6.3 Thesis Statement Revisited

The thesis statement from §1.2 is repeated in this section and subsequently followed by how this statement has been addressed.

> This thesis explores the different frameworks for implementing virtual network functions, including the programmable data plane, user space, and packet acceleration frameworks. **The thesis advocates advancing function virtualisation and deployment by exploiting diverse emerging environments, such as programmable switch hardware and in-network acceleration platforms, for VNF placement at the network edge.** Using extensive evaluations, key metrics such as end-to-end path latency, bandwidth, and resource utilization in the edge network environment demonstrated the promising benefits of leveraging heterogeneous packet processing elements in vNF deployments in next-generation edge networks to improve the overall QoS and SLA for subscribed users.

The beginning of this thesis described the advances in important networking paradigms such as NFV, SDN, SFC, and data plane programmability. The thesis motivated the need for network operators to leverage the inherent capabilities of heterogeneous packet processing elements as part of the service chain in deploying virtual network functions in next-generation networks.

The nature of virtual network functions and the heterogeneity of packet processing elements were investigated, providing insights into their diverse capabilities. A detailed investigation was also carried out, which will assist network operators with the important decision-making process of determining the right packet processing elements for deploying virtual network functions from diverse emerging use cases.

In addition to placing network functions at the network edge, closer to end users, the thesis also shows the benefits of leveraging a hybrid vNF deployment scheme, a major shift from the status quo of vNF placement, using the fundamentals of Integer Linear Programming. It presents the benefits that can be obtained by leveraging a hybrid vNF placement scheme, which includes minimizing end-to-end path latency in a dynamic edge network infrastructure, efficient resource utilization resulting from the optimal placement of virtual network functions on heterogeneous packet processing elements,

and meeting up with operator-defined SLAs in terms of application latency violations.

The dynamic nature of edge network environments also means that finding a way to make placement decisions quickly is necessary and comes with added benefits, as demonstrated in this thesis, using the proposed hybrid placement heuristic. The placement scheme presented in this thesis goes beyond showcasing the improved performance benefits for network operators, as it also caters to the underlying structure of service chains in emerging use cases, which often comes with a combination of latency-sensitive/tolerant applications, computationally intensive stateful functions, simple per packet operations, etc. Service providers can use this thesis's dynamic hybrid placement solution to cater to diverse edge placement use cases.

6.4 Further Use Cases

Some important real-life use cases that motivate the placement of network functions on heterogeneous packet processing elements are presented. Different classes of users request vNFs from the edge network infrastructure. Network operators can leverage the work presented in this thesis in these environments, which are generally characterized by diverse resource and SLA requirements. Some of these use cases are briefly explained below.

- **Industrial IoT**: Edge computing is well-suited for industrial IoT applications where low latency and real-time data processing are critical. By deploying edge devices at industrial sites, data can be processed locally, improving operational efficiency and predictive maintenance and enabling faster response times for critical events [171]. The edge computing architecture comes with the promise of low latency and high throughput; thus, this can be leveraged by industrial control systems, where low-latency applications are often deployed.

  To unlock the full potential of IIoT, it is essential to have an underlying network infrastructure that handles diverse applications. IIoT network infrastructures are often designed to handle various demands, ranging from ultra-reliable low-latency communications (URLLC) to latency-tolerant communications [172]. The work presented in this thesis fits in neatly with emerging IIoT use cases, which are characterised by having diverse latency and bandwidth requirements [173]; thus, network operators can adopt a framework that leverages the heterogeneity of edge hosts.

- **Smart Energy Management**: The growing adoption of smart grids for commercial electricity management has increased the number of related applications

used in power generation, transmission, distribution and smart metering. Applications used for load forecasting [174] or monitoring to detect energy theft [175] require high bandwidth and low latency to obtain real-time data from the smart grid infrastructure.

Edge computing can be vital in optimizing energy consumption and smart grid management [176], which often incorporates machine learning algorithms into the smart grid infrastructure. By deploying edge devices in smart grids and energy distribution networks, real-time monitoring, predictive analytics, and automated control systems, which are latency-sensitive, can be implemented on lightweight accelerated packet processing elements for efficient energy utilization and demand response [177].

Smart grid applications often come with tight QoS requirements in terms of rapid mobility and low latency, thus making quick response time a key component of the smart grid system [176]. Employing the edge computing paradigm improves the resilience of the smart grid infrastructure by ensuring high-throughput, low-latency communication between nodes in the smart grid edge infrastructure; thus, this can be extended by leveraging a hybrid vNF deployment framework for placing smart grid applications.

– **Mixed Reality (MR)**: Augmented Reality (AR) and Virtual Reality (VR) applications have increased over the past few years. This rise is due to the diverse applications of such technologies in healthcare, online gaming, education, sports, etc. [178]. These applications often come with various latency and bandwidth requirements, i.e., high bandwidth and ultra-low latency requirements.

  MR, which combines digital objects and real-life experiences for end users, is another emerging use case with minimal packet processing delay as a critical requirement [178]. MR applications and headsets require fast access to authentication gateways and content servers, with low latency to support these environments' high frames per second (FPS) requirements [179]. This is a suitable candidate for network operators to leverage the outcome of this thesis in ensuring that the underlying edge network infrastructure is provisioned with the correct packet processing elements along the service chain to serve diverse MR applications.

  In the MR use case, access control and authentication functionality could be offloaded onto lightweight programmable hosts, while cached MR content will be retrieved from user space hosts with abundant storage and I/O capabilities. This will also ensure that network operators can guarantee QoS and meet the SLAs of these applications for subscribed users.

– **Self-driving cars**: With the constant rise in driverless cars, one everyday use

case of edge vNF placement is self-driving cars [180]. Autonomous vehicles must dynamically process information received via onboard sensors to make key decisions, such as responding to traffic lights [181]. As it relates to the work presented in this thesis, on the one hand, network functions handling latency-sensitive tasks can be placed in the programmable data plane in the edge network infrastructure. On the other hand, operations requiring less crucial decisions, characterised as being more latency-tolerant, such as calculating complete travel routes, can be handled by network functions that accommodate much higher latency bounds and thus can be placed on user space hosts. Another important scenario related to autonomous driving is image recognition; thus, these applications, which are vital in identifying objects and obstructions, often involve using machine learning models [182]. This requires substantial processing capabilities (CPU, memory, and I/O) and can be placed on *user space hosts* along a packet processing pipeline.

– **E-healthcare**: A common edge computing use case is e-healthcare, which has seen tremendous growth as telesurgery is expected to grow in the coming years [183]. Everyday e-healthcare applications also facilitate essential tasks such as activity tracking, live monitoring, and constant measuring of calories, among other functions.

  One critical use case of the hybrid edge placement on heterogeneous hosts is performing basic tasks such as access control for retrieving patients' medical records from a secure location. This could be handled by a simple firewall network functionality implemented on a data plane packet processing element using fine-grained match-action rules.

  Medical equipment for performing time-sensitive operations, e.g., remote surgeries requiring robotic arms, which involve extreme care, precision, and high computation capacity, can be handled by network functions deployed in packet acceleration frameworks characterized by abundant computing resources.

6.5 Future Research

The research problems addressed in this thesis are still open-ended and attract the attention of researchers in this domain. The introductory part of this thesis (Chapters 1 and 2) captured some recent efforts and important related open-ended research challenges published in [3]. This thesis sets out to address some of the challenges identified, for example, the possible performance benefits of leveraging heterogeneous packet processing elements for NFV deployments [10] and the need for exploiting the

capabilities of diverse packet processing elements, including the programmable data plane as part of a service chain [170], there are still ways in which the work presented here can be extended. The work primarily focuses on how network operators can leverage the capability of heterogeneous packet processing elements to cater to the requirements of emerging edge computing use cases in future networks. Below are some possible ways of extending the work presented.

6.5.1 In-band Network Telemetry and Multi-domain Service Orchestration

To achieve end-to-end visualisation of the network and monitor performance, In-band network telemetry is used [184]. This comes with the advantage of acquiring real-time network packet measurements; unlike traditional network measurement techniques, in-band telemetry can easily leverage data plane programmability technologies such as P4 [185]. As presented in this thesis, attempting to extend programmability to the network edge requires diverse packet processing elements for deploying network functions. The work presented in this thesis can be extended to optimise the underlying infrastructure for implementing in-band telemetry solutions.

Efforts in the literature, such as the work by Sheng *et al.* [186], implemented an INT framework for network management using software and Tofino programmable switches, which achieved low bandwidth overhead. A recent use case that leverages data plane programmability for INT is HINT [187], which supports congestion control decisions using P4-driven INT. To achieve heterogeneity and high performance in INT, Hyun *et al.* [188] proposed an INT management system and an INT Collector, which leverages the capabilities of the eXpress Data Path (XDP) to achieve high performance and a custom event detection technique.

To leverage the capability of the programmable data plane for in-network telemetry, Bhamare *et al.* [185] presented IntOpt, which fetches telemetry data directly from the programmable data plane elements along a service chain. In the context of the end-to-end programmable continuum proposed in this thesis, efforts like IntOpt can be incorporated with other packet processing acceleration frameworks to introduce heterogeneity in in-network telemetry to minimise monitoring overhead.

In a similar context to achieving heterogeneity in INT, there is a need to consider the nature of future networks, i.e., the inevitable demand for service orchestration across multiple domains. Although NFV and network programmability also promise centralised management in edge network environments using controllers or other service orchestration algorithms. The reality is somewhat different, as the distributed nature of

next-generation edge IoT networks makes it challenging to achieve centralised control easily.

From the outcome of the literature review and the work presented in the thesis, the work can be extended to handle INT deployments using diverse VNF hosts, e.g., DPDK, containers, switch data plane and VMs. This is even more so as use cases for hybrid function placement are envisaged to span multiple connectivity domains such as non-terrestrial and terrestrial networks, resource-constrained industrial IoT implementations, etc. The use cases expected to emerge from implementing the 6G standard in the coming years will require novel mechanisms for the seamless orchestration and coordination of services across heterogeneous hosts in these domains.

The work presented in this thesis can also be extended in this direction since it has provided the foundation for exploiting the diverse capabilities of packet processing elements that constitute a service chain. One key future research objective is to design adaptive INT techniques that respond to placements based on edge network conditions, the type of network function, and the monitoring objective set out by network operators. Another research objective in this direction is to design an orchestration solution for hybrid VNF orchestration across multiple domains, which ensures optimal service delivery and resource utilisation. Developing protocols and standardised interfaces for interoperability across hybrid edge environments would allow for seamless distribution of telemetry data in the distributed edge environment.

6.5.2 Intelligent Hybrid Function Chaining at the Network Edge

Machine Learning and Artificial Intelligence applications are becoming popular, with more adoption expected to grow in the coming years. Regarding edge vNF placements, authors have combined Deep Reinforcement Learning with Graph Neural Networks to achieve the Optimal Placement of functions at the edge in an attempt to cater to diverse topologies [189]. Efforts such as the work by Deng *et al.* [190] used Deep Reinforcement Learning algorithms to select SFC paths to minimize latency between security functions.

The work by Khoshkholghi *et al.* [191] presented an online service chain deployment scheme to minimize the drop rate of service chain placement requests sent in by subscribed edge users. Similarly, Bunyaki [192] *et al.* employed Reinforcement Learning techniques tailored for traffic prediction by leveraging service-level performance predictions for vNF placements. Their proposal is geared towards achieving resilience in dynamic placement scenarios. Another recent approach in intelligent vNF placement

is the work by Yue *et al.* [193], which uses a Deep Learning-based function placement technique to select nodes to handle SFC requests efficiently; the primary goal of the proposal is to optimise resource utilisation.

The literature review has shown that efforts around intelligent VNF placement can be explored even further. A unique contribution to future research related to the work presented in this thesis is intelligently placing functions on heterogeneous hosts in the distributed edge network environment in a hybrid manner. In addition to the metrics considered in this thesis and in related literature, such as link bandwidth, latency, and the number of SLA violations by users connecting to their respective VNFs, additional constraints, such as energy utilisation and deployment cost optimisation, could benefit from using machine learning algorithms, such as Deep Reinforcement Learning algorithms, for leveraging packet processing elements' heterogeneity in a service chain.

Another unique extension to build upon the model presented involves 1) inferring the available resources on a diverse set of hosts along a processing pipeline in a distributed edge environment, i.e., taking into account the expected increasing diversity of SFCs in future networks, 2) selecting latency-optimal paths, w.r.t. defined SLA and service chain structure, i.e., designing frameworks that can be used in intelligently inferring, using available data, the possible structure of SFCs to allow for the provisioning of underlying heterogeneous network infrastructure, and 3) using network telemetry data to infer the nature and states network functions that constitute an SFC, amongst others. This would offer a novel perspective to hybrid edge VNF placement beyond what has been covered by related work.

6.6 Concluding Remarks

In recent years, network operators have significantly shifted towards network programmability through advances in NFV, SFC, and SDN, driven by the need to meet the diverse demands of subscribed users. Future networks are expected to embrace drastic changes, adopting end-to-end programmability beyond the data centre to the network's edge.

Since advances in software and hardware packet processing elements have brought about various (often lightweight) underlying devices and technologies capable of hosting and accelerating the performance of network functions, having a VNF framework that unifies diverse hosts to provide end-to-end connectivity along a packet processing pipeline is paramount.

This thesis demonstrated the hypothesis that in addition to placing network functions closer to end-users at the edge, using edge computing, NFV, and SFC, leveraging

diverse packet processing elements for placing network functions can help network operators cater to the requirements and SLAs of emerging use cases in future networks.

This thesis advocated advancing function virtualisation and deployment by exploiting network programmability over diverse emerging environments, such as programmable switch hardware and in-network acceleration platforms. Extensive evaluations supported the hypothesis that network operators can benefit immensely from adopting an end-to-end programmability continuum using heterogeneous packet processing elements to support emerging use cases.

The outcome of this thesis and similar research will change future networks in some fundamental ways. This will revolutionise network service delivery by paving the way for a new paradigm that leverages edge vNF placement's full potential. Adopting a framework that unifies diverse packet processing elements will allow future networks to provide faster and more flexible services to meet the emerging requirements of applications and users.

I also see the findings presented in this thesis that enable network operators to design future edge networks that are dynamic and scalable, i.e., not bound by the restrictions/capabilities of some underlying packet processing elements that constitute a service chain. This will also empower network operators to design networks that respond efficiently to unpredictable workloads; thus, use cases that require high bandwidth for data-intensive operations or low latency for real-time packet processing can be efficiently handled.

I envision adopting end-to-end programmability along a service chain, using heterogeneous packet processing elements as a *silver bullet* that will cater to many emerging edge use cases. As a real-life example, a recent study by Delloite [194] projected the future of healthcare in 2040, which is believed to be driven by data and platforms, in what will be known simply as *consumer-driven health*. This will involve using diverse e-healthcare edge devices that require constant, resilient and secure connectivity to edge data centres, often running Machine Learning and Artificial Intelligence models on lightweight edge devices to provide personalised healthcare solutions. The work presented in this thesis and similar efforts in the literature will serve as an enabler for handling such use cases by network operators.

With the recent rise in the number of compute-intensive, ultra-low latency applications with AI and ML capabilities, which is expected to grow even further in coming years, adopting an end-to-end programmability continuum, as advocated in this thesis, paves the way for further innovation in the design of future networks, as it will create the proper underlying network infrastructure for novel technologies and applications to thrive.

6.7 Publications

The work presented in this thesis led to the following publications:

1. Haruna Umar Adoga and Dimitrios. P. Pezaros, "Optimal Edge vNF Placements Leveraging Diverse Programmable Platforms" – **under submission to IEEE TNSM**
2. Haruna Umar Adoga and Dimitrios P. Pezaros, "Towards latency-aware vnf placement on heterogeneous hosts at the network edge," in 2023 IEEE Global Communications Conference (GLOBECOM), 2023. [170]
3. Haruna Umar Adoga, Yehia. Elkhatib, and Dimitiros. P. Pezaros, "On the performance benefits of heterogeneous virtual network function execution frameworks," in 2022 IEEE 8th International Conference on Network Softwarization (NetSoft). IEEE, 2022, pp. 109–114 [10]
4. Haruna Umar Adoga and Dimitrios. P. Pezaros, "Network function virtualisation and service function chaining frameworks: A comprehensive review of requirements, objectives, implementations, and open research challenges," Future Internet, vol. 14, no. 2, p. 59, 2022 [3]

Bibliography

[1] Ericson, "Why iot changes everything," *Last Accessed: October*, 2023. [Online]. Available: https://www.ericsson.com/en/internet-of-things

[2] K. A. Simpson, "Online learning on the programmable dataplane," Ph.D. dissertation, University of Glasgow, 2022.

[3] H. U. Adoga and D. P. Pezaros, "Network function virtualization and service function chaining frameworks: A comprehensive review of requirements, objectives, implementations, and open research challenges," *Future Internet*, vol. 14, no. 2, p. 59, 2022.

[4] J. G. Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.

[5] S. Cherrared, S. Imadali, E. Fabre, G. Gössler, and I. G. B. Yahia, "A survey of fault management in network virtualization environments: Challenges and solutions," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1537–1551, 2019.

[6] F. Paganelli, P. Cappanera, and G. Cuffaro, "Tenant-defined service function chaining in a multi-site network slice," *Future Generation Computer Systems*, vol. 121, pp. 1–18, 2021.

[7] X. Li and C. Qian, "A survey of network function placement," in *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2016, pp. 948–953.

[8] E. Haleplidis, D. Joachimpillai, J. H. Salim, D. Lopez, J. Martin, K. Pentikousis, S. Denazis, and O. Koufopavlou, "Forces applicability to sdn-enhanced nfv," in *2014 Third European Workshop on Software Defined Networks*. IEEE, 2014, pp. 43–48.

[9] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Pro-

gramming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[10] H. U. Adoga, Y. Elkhatib, and D. P. Pezaros, "On the performance benefits of heterogeneous virtual network function execution frameworks," in *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*. IEEE, 2022, pp. 109–114.

[11] K. A. Simpson, R. Cziva, and D. P. Pezaros, "Seir: Dataplane assisted flow classification using ml," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.

[12] D. Kim, Y. Zhu, C. Kim, J. Lee, and S. Seshan, "Generic external memory for switch data planes," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, 2018, pp. 1–7.

[13] K. N. Qureshi, E. Ahmad, M. Anwar, K. Z. Ghafoor, and G. Jeon, "Network functions virtualization for mobile core and heterogeneous cellular networks," *Wireless Personal Communications*, vol. 122, no. 3, pp. 2543–2559, 2022.

[14] Z. Huang, N. Samaan, and A. Karmouch, "A novel resource reliability-aware infrastructure manager for containerized network functions," in *International Conference on Communications (ICC)*. IEEE, 2021.

[15] X. Wang, J. Du, and H. Liu, "Performance and isolation analysis of runc, gvisor and kata containers runtimes," *Cluster Computing*, pp. 1–17, 2022.

[16] W. Zhang, G. Liu, W. Zhang, N. Shah, P. Lopreiato, G. Todeschi, K. Ramakrishnan, and T. Wood, "Opennetvm: A platform for high performance network service chains," in *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*, 2016, pp. 26–31.

[17] L. Abeni, "Real-time unikernels: A first look," in *International Conference on High Performance Computing*. Springer, 2023, pp. 121–133.

[18] R. Cziva, S. Jouet, K. J. White, and D. P. Pezaros, "Container-based network function virtualization for software-defined networks," in *2015 IEEE symposium on computers and communication (ISCC)*. IEEE, 2015, pp. 415–420.

[19] V. Aggarwal and B. Thangaraju, "Performance analysis of virtualisation technologies in nfv and edge deployments," in *2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*. IEEE, 2020, pp. 1–5.

[20] F. Paolucci, D. Scano, P. Castoldi, and E. De Paoli, "Latency control in service chaining using p4-based data plane programmability," *Computer Networks*, vol. 216, p. 109227, 2022.

[21] J. Kim, Y. Kim, V. Yegneswaran, P. Porras, S. Shin, and T. Park, "Extended data plane architecture for in-network security services in software-defined networks," *Computers & Security*, p. 102976, 2022.

[22] N. Sultana, J. Sonchack, H. Giesen, I. Pedisich, Z. Han, N. Shyamkumar, S. Burad, A. DeHon, and B. T. Loo, "Flightplan: Dataplane disaggregation and placement for p4 programs," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 571–592.

[23] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "Pint: Probabilistic in-band network telemetry," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 662–680.

[24] X. Li, X. Wang, F. Liu, and H. Xu, "Dhl: Enabling flexible software network functions with fpga acceleration," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 1–11.

[25] T.-H. Yen and C.-S. Yang, "Flexips: A keep tracking scalable network function design and implementation," in *2021 2nd International Conference on Electronics, Communications and Information Technology (CECIT)*. IEEE, 2021, pp. 607–613.

[26] M. S. Castanho, C. K. Dominicini, R. S. Villacça, M. Martinello, and R. M. Ribeiro, "Phantomsfc: A fully virtualized and agnostic service function chaining architecture," in *2018 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2018, pp. 354–359.

[27] L. Lai, G. Ara, T. Cucinotta, K. Kondepu, and L. Valcarenghi, "Ultra-low latency nfv services using dpdk," in *2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2021, pp. 8–14.

[28] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines, "5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges," *Computer Networks*, vol. 167, p. 106984, 2020.

[29] D. E. Sarmiento, A. Lebre, L. Nussbaum, and A. Chari, "Decentralized sdn control plane for a distributed cloud-edge infrastructure: A survey," *IEEE Communications Surveys & Tutorials*, 2021.

[30] W. Tu, Y.-H. Wei, G. Antichi, and B. Pfaff, "Revisiting the open vSwitch dataplane ten years later," in *ACM SIGCOMM Conference*, 2021, pp. 245–257.

[31] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*. IEEE, 2014, pp. 1–6.

[32] S. Kaur, J. Singh, and N. S. Ghumman, "Network programmability using pox controller," in *Proc. Int. Conf. Commun., Comput. Syst.(ICCCS)*, vol. 138, 2014, pp. 134–138.

[33] RYU, "What is RYU?" https://ryu-sdn.org/, Accessed 2023, last assessed on May 5, 2023.

[34] T. Zhang, L. Linguaglossa, P. Giaccone, L. Iannone, and J. Roberts, "Performance benchmarking of state-of-the-art software switches for nfv," *Computer Networks*, vol. 188, p. 107861, 2021.

[35] L. S. Peter, H. Kobo, and V. M. Srivastava, "A comparative review analysis of openflow and p4 protocols based on software defined networks," *Data Intelligence and Cognitive Informatics: Proceedings of ICDICI 2022*, pp. 699–711, 2022.

[36] A. M. Medhat, G. A. Carella, M. Pauls, M. Monachesi, M. Corici, and T. Magedanz, "Resilient orchestration of service functions chains in a nfv environment," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2016, pp. 7–12.

[37] C. Cascone, "P4 support in onos," *Proc. ONOS Build*, pp. 1–29, 2017.

[38] F. Cugini, D. Scano, A. Giorgetti, A. Sgambelluri, P. Castoldi, and F. Paolucci, "P4 programmability at the network edge: the braine approach," in *2021 International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2021, pp. 1–9.

[39] A. Al Sadi, M. Savi, D. Berardi, A. Melis, M. Prandini, and F. Callegati, "Real-time pipeline reconfiguration of p4 programmable switches to efficiently detect and mitigate ddos attacks," in *2023 26th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. IEEE, 2023, pp. 21–23.

[40] K. A. Simpson, C. Williamson, D. J. Paul, and D. P. Pezaros, "Galette: a lightweight xdp dataplane on your raspberry pi," in *2023 IFIP Networking Conference (IFIP Networking)*. IEEE, 2023, pp. 1–9.

[41] 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.

[42] S. Ibanez, G. Brebner, N. McKeown, and N. Zilberman, "The p4-¿ netfpga workflow for line-rate packet processing," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 1–9.

[43] M. Shahbaz, S. Choi, B. Pfaff, C. Kim, N. Feamster, N. McKeown, and J. Rexford, "Pisces: A programmable, protocol-independent software switch," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 525–538.

[44] Ericson, "Behavioral model (bmv2): The reference p4 software switch," *Last Accessed: October*, 2016. [Online]. Available: https://github.com/p4lang/behavioral-model

[45] T. Zhang, H. Qiu, L. Linguaglossa, W. Cerroni, and P. Giaccone, "Nfv platforms: Taxonomy, design choices and future challenges," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 30–48, 2020.

[46] ETSI, "Network functions virtualisation (NFV): Architectural framework," Tech. Rep. 002 V1.1.1, Oct. 2013.

[47] A. Binu and G. S. Kumar, "Virtualization techniques: a methodical review of xen and kvm," in *International Conference on Advances in Computing and Communications*. Springer, 2011, pp. 399–410.

[48] M. J. Wray and C. I. Dalton, "Network virtualization," Jul. 17 2012, uS Patent 8,223,770.

[49] R. Ricart-Sanchez, P. Malagon, P. Salva-Garcia, E. C. Perez, Q. Wang, and J. M. A. Calero, "Towards an fpga-accelerated programmable data path for edge-to-core communications in 5g networks," *Journal of Network and Computer Applications*, vol. 124, pp. 80–93, 2018.

[50] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz, "Service function chaining in next generation networks: State of the art and research challenges," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 216–223, 2016.

[51] F. Asquini, A. Bujari, D. Munaretto, C. E. Palazzi, and D. Ronzani, "An ETSI NFV implementation for automatic deployment and configuration of a virtualized mobile core network," in *International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2021, pp. 357–362.

[52] X. Zhang, L. Cui, F. P. Tso, Z. Li, and W. Jia, "Dapper: Deploying service function chains in the programmable data plane via deep reinforcement learning," *IEEE Transactions on Services Computing*, 2023.

[53] R. Kundel, L. Nobach, J. Blendin, W. Maas, A. Zimber, H.-J. Kolbe, G. Schyguda, V. Gurevich, R. Hark, B. Koldehofe *et al.*, "Openbng: Central office network functions on programmable data plane hardware," *International Journal of Network Management*, vol. 31, no. 1, p. e2134, 2021.

[54] S. K. Singh, C. E. Rothenberg, G. Patra, and G. Pongracz, "Offloading virtual evolved packet gateway user plane functions to a programmable asic," in *Proceedings of the 1st ACM CoNEXT Workshop on Emerging in-Network Computing Paradigms*, 2019, pp. 9–14.

[55] K. Neshatpour, M. Malik, M. A. Ghodrat, A. Sasan, and H. Homayoun, "Energy-efficient acceleration of big data analytics applications using fpgas," in *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, 2015, pp. 115–123.

[56] A. Tulumello, G. Belocchi, M. Bonola, S. Pontarelli, and G. Bianchi, "Pushing services to the edge using a stateful programmable dataplane," in *2019 European Conference on Networks and Communications (EuCNC)*. IEEE, 2019, pp. 389–393.

[57] M. Wang, B. Cheng, S. Wang, and J. Chen, "Availability-and traffic-aware placement of parallelized sfc in data center networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 182–194, 2021.

[58] Y. Li and M. Chen, "Software-defined network function virtualization: A survey," *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[59] P. Grønsund, K. Mahmood, G. Millstein, A. Noy, G. Solomon, and A. Sahai, "A solution for sgi-lan services virtualization using nfv and sdn," in *2015 European Conference on Networks and Communications (EuCNC)*. IEEE, 2015, pp. 408–412.

[60] F. Z. Yousaf, M. Bredel, S. Schaller, and F. Schneider, "Nfv and sdn—key technology enablers for 5g networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2468–2478, 2017.

[61] A. Kak, "Towards 6g through sdn and nfv-based solutions for terrestrial and non-terrestrial networks," Ph.D. dissertation, Georgia Institute of Technology, 2021.

[62] Y. A. Qadri, A. Nauman, Y. B. Zikria, A. V. Vasilakos, and S. W. Kim, "The future of healthcare internet of things: a survey of emerging technologies," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1121–1167, 2020.

[63] M. Huang, A. Liu, N. N. Xiong, T. Wang, and A. V. Vasilakos, "An effective service-oriented networking management architecture for 5g-enabled internet of things," *Computer networks*, vol. 173, p. 107208, 2020.

[64] M. E. Morocho-Cayamcela, H. Lee, and W. Lim, "Machine learning for 5g/b5g mobile and wireless communications: Potential, limitations, and future directions," *IEEE Access*, vol. 7, pp. 137 184–137 206, 2019.

[65] G. Berardinelli, N. H. Mahmood, I. Rodriguez, and P. Mogensen, "Beyond 5g wireless irt for industry 4.0: Design principles and spectrum aspects," in *2018 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2018, pp. 1–6.

[66] M. Katz, M. Matinmikko-Blue, and M. Latva-Aho, "6genesis flagship program: Building the bridges towards 6g-enabled wireless smart society and ecosystem," in *2018 IEEE 10th Latin-American Conference on Communications (LATINCOM)*. IEEE, 2018, pp. 1–9.

[67] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Network function virtualization in 5g," *IEEE Communications Magazine*, vol. 54, no. 4, pp. 84–91, 2016.

[68] Y. Liu, F. Zhou, C. Chen, Z. Zhu, T. Shang, and J.-M. Torres-Moreno, "Disaster protection in inter-datacenter networks leveraging cooperative storage," *IEEE Transactions on Network and Service Management*, 2021.

[69] X. Zhong, Y. Wang, and X. Qiu, "Service function chain orchestration across multiple clouds," *China Communications*, vol. 15, no. 10, pp. 99–116, 2018.

[70] V. A. Cunha, I. D. Cardoso, J. P. Barraca, and R. L. Aguiar, "Policy-driven vcpe through dynamic network service function chaining," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. IEEE, 2016, pp. 156–160.

[71] M. Shojafar, Z. Pooranian, M. Sookhak, and R. Buyya, "Recent advances in cloud data centers toward fog data centers," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 8, p. e5164, 2019.

[72] Z. Yan, P. Zhang, and A. V. Vasilakos, "A security and trust framework for virtualized networks and software-defined networking," *Security and communication networks*, vol. 9, no. 16, pp. 3059–3069, 2016.

[73] H. Huang, C. Zeng, Y. Zhao, G. Min, Y. Y. Zhu, W. Miao, and J. Hu, "Scalable service function chain orchestration in nfv-enabled networks: A federated reinforcement learning approach," *IEEE Journal on Selected Areas in Communications*, 2021.

[74] G. Sun, Z. Xu, H. Yu, X. Chen, V. Chang, and A. V. Vasilakos, "Low-latency and resource-efficient service function chaining orchestration in network function virtualization," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5760–5772, 2019.

[75] A. Bremler-Barr, Y. Harchol, and D. Hay, "Openbox: a software-defined framework for developing, deploying, and managing network functions," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 511–524.

[76] G. P. Katsikas, M. Enguehard, M. Kuźniar, G. Q. Maguire Jr, and D. Kostić, "Snf: Synthesizing high performance nfv service chains," *PeerJ Computer Science*, vol. 2, p. e98, 2016.

[77] G. P. Katsikas, T. Barbette, D. Kostić, J. G. Q. Maguire, and R. Steinert, "Metron: High-performance nfv service chaining even in the presence of black-boxes," *ACM Transactions on Computer Systems (TOCS)*, vol. 38, no. 1-2, pp. 1–45, 2021.

[78] Y. He, X. Zhang, Z. Xia, Y. Liu, K. Sood, and S. Yu, "Joint optimization of service chain graph design and mapping in nfv-enabled networks," *Computer Networks*, vol. 202, p. 108626, 2022.

[79] D. R. Mafioletti, C. K. Dominicini, M. Martinello, R. M. Ribeiro, and R. d. S. Villaca, "Piaffe: A place-as-you-go in-network framework for flexible embedding of vnfs," *IEEE International Conference on Communications*, 2020.

[80] B. Anwer, T. Benson, N. Feamster, and D. Levin, "Programming slick network functions," in *Proceedings of the 1st acm sigcomm symposium on software defined networking research*, 2015, pp. 1–13.

[81] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, 2014, pp. 459–473.

[82] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker, "Netbricks: Taking the v out of {NFV}," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 203–216.

[83] Z. Meng, J. Bi, H. Wang, C. Sun, and H. Hu, "Coco: Compact and optimized consolidation of modularized service function chains in nfv," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–7.

[84] L. Li, K. Ota, and M. Dong, "Deepnfv: A lightweight framework for intelligent edge network functions virtualization," *IEEE Network*, vol. 33, no. 1, pp. 136–141, 2018.

[85] A. Lombardo, A. Manzalini, G. Schembra, G. Faraci, C. Rametta, and V. Riccobene, "An open framework to enable netfate (network functions at the edge)," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2015, pp. 1–6.

[86] K. Yasukata, F. Huici, V. Maffione, G. Lettieri, and M. Honda, "Hypernf: Building a high performance, high utilization and fair nfv platform," in *Proceedings of the 2017 Symposium on Cloud Computing*, 2017, pp. 157–169.

[87] C. Zheng, Q. Lu, J. Li, Q. Liu, and B. Fang, "A flexible and efficient container-based nfv platform for middlebox networking," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, 2018, pp. 989–995.

[88] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein, "Maglev: A fast and reliable software network load balancer," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 523–535.

[89] M. Ghaznavi, E. Jalalpour, B. Wong, R. Boutaba, and A. J. Mashtizadeh, "Fault tolerant service function chaining," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 198–210.

[90] L. Wang, W. Mao, J. Zhao, and Y. Xu, "Ddqp: A double deep q-learning approach to online fault-tolerant sfc placement," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 118–132, 2021.

[91] B. Dab, I. Fajjari, M. Rohon, C. Auboin, and A. Diquélou, "An efficient traffic steering for cloud-native service function chaining," in *2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. IEEE, 2020, pp. 71–78.

[92] S. G. Kulkarni, G. Liu, K. Ramakrishnan, M. Arumaithurai, T. Wood, and X. Fu, "Reinforce: Achieving efficient failure resiliency for network function virtualization-based services," *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, pp. 695–708, 2020.

[93] A. Hmaity, M. Savi, F. Musumeci, M. Tornatore, and A. Pattavina, "Virtual network function placement for resilient service chain provisioning," in *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*. IEEE, 2016, pp. 245–252.

[94] Y. Harchol, D. Hay, and T. Orenstein, "Ftvnf: Fault tolerant virtual network functions," in *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems*, 2018, pp. 141–147.

[95] H. B. Nguyen, N.-T. Dinh, J. Oh, and Y. Kim, "An openflow-based scheme for service chaining's high availability in cloud network," *International Conference on ICT Convergence*, 2019.

[96] S. Cai, F. Zhou, Z. Zhang, and A. Meddahi, "Disaster-resilient service function chain embedding based on multi-path routing," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2021, pp. 1–7.

[97] G. Mirjalily and L. Zhiquan, "Optimal network function virtualization and service function chaining: A survey," *Chinese Journal of Electronics*, vol. 27, no. 4, pp. 704–717, 2018.

[98] M. Shahjalal, N. Farhana, P. Roy, and M. A. Razzaque, "Qoe aware optimal deployment of virtual network functions in 5g hybrid cloud," in *2021 3rd International Conference on Sustainable Technologies for Industry 4.0 (STI)*. IEEE, 2021, pp. 1–6.

[99] M. M. Iordache-Sica, C. Anagnostopoulos, and D. P. Pezaros, "Towards qos-aware provisioning of chained virtual security services in edge networks," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2021, pp. 178–186.

[100] C. Chen, L. Nagel, L. Cui, and F. P. Tso, "Distributed federated service chaining for heterogeneous network environments," in *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing*, 2021, pp. 1–10.

[101] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic, latency-optimal vnf placement at the network edge," in *Ieee infocom 2018-ieee conference on computer communications*. IEEE, 2018, pp. 693–701.

[102] A. Ali, C. Anagnostopoulos, and D. P. Pezaros, "In-network placement of security vnfs in multi-tenant data centers," in *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2020, pp. 1–6.

[103] F. Carpio, W. Bziuk, and A. Jukan, "On optimal placement of hybrid service function chains (sfcs) of virtual machines and containers in a generic edge-cloud continuum," *arXiv preprint arXiv:2007.04151*, 2020.

[104] T. Moufakir, M. F. Zhani, A. Gherbi, M. Aloqaily, and N. Ghrada, "Sfcaas: Service function chains as a service in nfv environments," *arXiv preprint arXiv:2203.01098*, 2022.

[105] J. Halpern, C. Pignataro *et al.*, "Service function chaining (sfc) architecture," in *RFC 7665*. IETF, 2015, pp. 0–28.

[106] M. Özdem and M. Alkan, "Subscriber aware dynamic service function chaining," *Computer Networks*, vol. 194, p. 108138, 2021.

[107] S. Aidi, M. F. Zhani, and Y. Elkhatib, "On optimizing backup sharing through efficient vnf migration," in *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 60–65.

[108] H. Zhu and C. Gehrmann, "Lic-sec: an enhanced apparmor docker security profile generator," *Journal of Information Security and Applications*, vol. 61, p. 102924, 2021.

[109] J. L. Chen, D. Liaqat, M. Gabel, and E. de Lara, "Starlight: Fast container provisioning on the edge and over the {WAN}," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 35–50.

[110] A. Stockmayer, S. Hinselmann, M. Häberle, and M. Menth, "Service function chaining based on segment routing using p4 and sr-iov (p4-sfc)," in *High Performance Computing: ISC High Performance 2020 International Workshops, Frankfurt, Germany, June 21–25, 2020, Revised Selected Papers 35*. Springer, 2020, pp. 297–309.

[111] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, "The express data path: Fast programmable packet processing in the operating system kernel," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, 2018, pp. 54–66.

[112] X. Lin, Y. Chen, X. Li, J. Mao, J. He, W. Xu, and Y. Shi, "Scalable kernel tcp design and implementation for short-lived connections," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 2, pp. 339–352, 2016.

[113] F. Parola, R. Procopio, R. Querio, and F. Risso, "Comparing user space and in-kernel packet processing for edge data centers," *ACM SIGCOMM Computer Communication Review*, vol. 53, no. 1, pp. 14–29, 2023.

[114] S. Sharma, N. Uniyal, B. Tola, and Y. Jiang, "On monolithic and microservice deployment of network functions," in *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 387–395.

[115] S. R. Chowdhury, M. A. Salahuddin, N. Limam, and R. Boutaba, "Re-architecting nfv ecosystem with microservices: State of the art and research challenges," *IEEE Network*, vol. 33, no. 3, pp. 168–176, 2019.

[116] Z. Meng, J. Bi, H. Wang, C. Sun, and H. Hu, "Micronf: An efficient framework for enabling modularized service chains in nfv," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 8, pp. 1851–1865, 2019.

[117] S. G. Kulkarni, G. Liu, K. Ramakrishnan, M. Arumaithurai, T. Wood, and X. Fu, "Reinforce: Achieving efficient failure resiliency for network function virtualization based services," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, 2018, pp. 41–53.

[118] D. Perino, M. Varvello, L. Linguaglossa, R. Laufer, and R. Boislaigue, "Caesar: A content router for high-speed forwarding on content names," in *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2014, pp. 137–148.

[119] X. Chen, H. Liu, D. Zhang, Z. Meng, Q. Huang, H. Zhou, C. Wu, X. Liu, and Q. Yang, "Automatic performance-optimal offloading of network functions on programmable switches," *IEEE Transactions on Cloud Computing*, 2022.

[120] G. P. Katsikas, T. Barbette, D. Kostic, R. Steinert, and G. Q. Maguire Jr, "Metron:{NFV} service chains at the true speed of the underlying hardware," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, 2018, pp. 171–186.

[121] D. Saxena and A. K. Singh, "OFP-TM: an online VM failure prediction and tolerance model towards high availability of cloud computing environments," *The Journal of Supercomputing*, pp. 1–22, 2022.

[122] A. Bouridah, I. Fajjari, N. Aitsaadi, and H. Belhadef, "Optimized scalable sfc traffic steering scheme for cloud native based applications," in *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2021, pp. 1–6.

[123] S. Natarajan, R. Krishnan, A. Ghanwani, D. Krishnaswamy, P. Willis, A. Chaudhary, and F. Huici, "An analysis of lightweight virtualization technologies for nfv," in *Proc. IETF Draft*, 2017, pp. 1–16.

[124] E. J. Jackson, M. Walls, A. Panda, J. Pettit, B. Pfaff, J. Rajahalme, T. Koponen, and S. Shenker, "SoftFlow: A middlebox architecture for Open vSwitch," in *USENIX Annual Technical Conference (ATC 16)*, 2016, pp. 15–28.

[125] T. Kurek, "Unikernel network functions: A journey beyond the containers," *IEEE Communications Magazine*, vol. 57, no. 12, pp. 15–19, 2019.

[126] A. Laghrissi, T. Taleb, M. Bagaa, and H. Flinck, "Towards edge slicing: VNF placement algorithms for a dynamic & realistic edge cloud environment," in *IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2017, pp. 1–6.

[127] G. S. Niemiec, L. M. Batista, A. E. Schaeffer-Filho, and G. L. Nazar, "A survey on fpga support for the feasible execution of virtualized network functions," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 504–525, 2019.

[128] S. Qi, Z. Zeng, L. Monis, and K. Ramakrishnan, "Middlenet: A unified, high-performance nfv and middlebox framework with ebpf and dpdk," *IEEE Transactions on Network and Service Management*, 2023.

[129] D. Medhi and K. Ramasamy, "Chapter 12–routing and traffic engineering in data center networks," *Network Routing (Second Edition), second edition edn. The Morgan Kaufmann Series in Networking, Morgan Kaufmann, Boston*, pp. 396–422, 2018.

[130] B. Pfaff, "Open vSwitch manual," *Manual, Open vSwitch,[Accessed 10 October, 2015]*.

[131] A. Arins, "Firewall as a service in sdn openflow network," in *2015 IEEE 3rd Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*. IEEE, 2015, pp. 1–5.

[132] P. Krongbaramee and Y. Somchit, "Implementation of sdn stateful firewall on data plane using open vswitch," in *2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE)*. IEEE, 2018, pp. 1–5.

[133] J. Pettit and T. Graf, "Stateful connection tracking & stateful nat," in *Open vSwitch Fall Conference*, 2014.

[134] P. N. Ayuso, "The conntrack-tools user manual," 2008.

[135] T. A. Hoff, "Extending open vswitch to faciliate creation of stateful sdn applications," 2018.

[136] S. Sagkriotis and D. Pezaros, "Scale-friendly in-network coordination," in *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, 2022, pp. 5747–5752.

[137] H. Lefeuvre, G. Gain, D. Dinca, A. Jung, S. Kuenzer, V.-A. Badoiu, R. Deaconescu, L. Mathy, C. Raiciu, P. Olivier *et al.*, "Unikraft and the coming of age of unikernels," *login; The Usenix Magazine*, 2021.

[138] J. B. Filipe, F. Meneses, A. Rehman, D. Corujo, and R. L. Aguiar, "A performance comparison of containers and unikernels for reliable 5g environments," in *2019 15th International Conference on the Design of Reliable Communication Networks (DRCN)*. IEEE, 2019, pp. 99–106.

[139] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, 2000.

[140] Xilinx. (Accessed 2022) Accelerator Cards. https://www.xilinx.com/products/boards-and-kits/accelerator-cards.html. Last assessed on March 5, 2022.

[141] A. Rivitti, R. Bifulco, A. Tulumello, M. Bonola, and S. Pontarelli, "ehdl: Turning ebpf/xdp programs into hardware designs for the nic," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023, pp. 208–223.

[142] B. Li, K. Tan, L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen, "Clicknp: Highly flexible and high performance network processing with reconfigurable hardware," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 1–14.

[143] S. McCanne and V. Jacobson, "The bsd packet filter: A new architecture for user-level packet capture." in *USENIX winter*, vol. 46, 1993, pp. 259–270.

[144] M. Fleming. (2017) A thorough introduction to ebpf. Accessed on: 10/10/23. [Online]. Available: https://lwn.net/Articles/740157/

[145] N. Van Tu, J.-H. Yoo, and J. W.-K. Hong, "Building hybrid virtual network functions with express data path," in *2019 15th International Conference on Network and Service Management (CNSM)*. IEEE, 2019, pp. 1–9.

[146] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *Proceedings of the Symposium on SDN Research*, 2016, pp. 1–12.

[147] G. Grigoryan, Y. Liu, and M. Kwon, "iload: In-network load balancing with programmable data plane," in *Proceedings of the 15th International Conference on emerging Networking EXperiments and Technologies*, 2019, pp. 17–19.

[148] M. Kulkarni, B. Goswami, and J. Paulose, "Hula: Dynamic and scalable load balancing mechanism for data plane of sdn," in *2023 Fifth International Conference on Electrical, Computer and Communication Technologies (ICECCT)*. IEEE, 2023, pp. 1–9.

[149] K. A. Simpson and D. P. Pezaros, "Online rl in the programmable dataplane with opal," in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, 2021, pp. 471–472.

[150] C. Sun, J. Bi, Z. Zheng, and H. Hu, "Hyper: A hybrid high-performance framework for network function virtualization," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2490–2500, 2017.

[151] L. d. C. Marcuzzo and C. R. dos Santos, "Enabling partial offload of virtualized network functions into the programmable data plane," in *2020 IEEE Latin-*

American Conference on Communications (LATINCOM). IEEE, 2020, pp. 1–6.

[152] N. Van Tu, K. Ko, and J. W.-K. Hong, "Architecture for building hybrid kernel-user space virtual network functions," in *2017 13th International Conference on Network and Service Management (CNSM)*. IEEE, 2017, pp. 1–6.

[153] H. Lu, N. Arora, H. Zhang, C. Lumezanu, J. Rhee, and G. Jiang, "Hybnet: Network manager for a hybrid network infrastructure," in *Proceedings of the Industrial Track of the 13th ACM/IFIP/USENIX International Middleware Conference*, 2013, pp. 1–6.

[154] J. Cao, Y. Zhang, W. An, X. Chen, Y. Han, and J. Sun, "Vnf placement in hybrid nfv environment: Modeling and genetic algorithms," in *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2016, pp. 769–777.

[155] H. Moens and F. De Turck, "Vnf-p: A model for efficient placement of virtualized network functions," in *10th international conference on network and service management (CNSM) and workshop*. IEEE, 2014, pp. 418–423.

[156] H. Huang, S. Guo, J. Wu, and J. Li, "Service chaining for hybrid network function," *IEEE Transactions on Cloud Computing*, vol. 7, no. 4, pp. 1082–1094, 2017.

[157] A. Leivadeas, G. Kesidis, M. Ibnkahla, and I. Lambadaris, "Vnf placement optimization at the edge and cloud," *Future Internet*, vol. 11, no. 3, p. 69, 2019.

[158] P. M. Mohan and M. Gurusamy, "Resilient vnf placement for service chain embedding in diversified 5g network slices," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[159] K. Kaur, V. Mangat, and K. Kumar, "A review on virtualized infrastructure managers with management and orchestration features in nfv architecture," *Computer Networks*, vol. 217, p. 109281, 2022.

[160] G. A. Carella and T. Magedanz, "Open baton: a framework for virtual network function management and orchestration for emerging software-based 5g networks," *Newsletter*, vol. 2016, 2015.

[161] L. A. Haibeh, M. C. Yagoub, and A. Jarray, "A survey on mobile edge computing infrastructure: Design, resource management, and optimization approaches," *IEEE Access*, vol. 10, pp. 27 591–27 610, 2022.

[162] H. Chen, R. Abbas, P. Cheng, M. Shirvanimoghaddam, W. Hardjawana, W. Bao, Y. Li, and B. Vucetic, "Ultra-reliable low latency cellular networks: Use cases,

challenges and approaches," *IEEE Communications Magazine*, vol. 56, no. 12, pp. 119–125, 2018.

[163] M. Ahmad, S. U. Jafri, A. Ikram, W. N. A. Qasmi, M. A. Nawazish, Z. A. Uzmi, and Z. A. Qazi, "A low latency and consistent cellular control plane," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 648–661.

[164] Q.-D. Ho, Y. Gao, and T. Le-Ngoc, "Challenges and research opportunities in wireless communication networks for smart grid," *IEEE Wireless Communications*, vol. 20, no. 3, pp. 89–95, 2013.

[165] Y. Rafique, A. Leivadeas, and M. Ibnkahla, "An iot-aware vnf placement proof of concept in a hybrid edge-cloud smart city environment," in *2022 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2022, pp. 1395–1400.

[166] I. Alghamdi, "Computation offloading in mobile edge computing: an optimal stopping theory approach," Ph.D. dissertation, University of Glasgow, 2021.

[167] T. S. Ferguson. Optimal stopping and applications.

[168] I. Leyva-Pupo, C. Cervelló-Pastor, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic scheduling and optimal reconfiguration of upf placement in 5g networks," in *Proceedings of the 23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2020, pp. 103–111.

[169] K. Cao, Y. Liu, G. Meng, and Q. Sun, "An overview on edge computing research," *IEEE access*, vol. 8, pp. 85 714–85 728, 2020.

[170] H. U. Adoga and D. P. Pezaros, "Towards latency-aware vnf placement on heterogeneous hosts at the network edge," in *GLOBECOM 2023-2023 IEEE Global Communications Conference*. IEEE, 2023, pp. 6383–6388.

[171] M. Dangana, S. Ansari, Q. H. Abbasi, S. Hussain, and M. A. Imran, "Suitability of nb-iot for indoor industrial environment: A survey and insights," *Sensors*, vol. 21, no. 16, p. 5284, 2021.

[172] A. Narayanan, A. S. De Sena, D. Gutierrez-Rojas, D. C. Melgarejo, H. M. Hussain, M. Ullah, S. Bayhan, and P. H. Nardelli, "Key advances in pervasive edge computing for industrial internet of things in 5g and beyond," *IEEE Access*, vol. 8, pp. 206 734–206 754, 2020.

[173] M. Alabadi, A. Habbal, and X. Wei, "Industrial internet of things: Requirements, architecture, challenges, and future research directions," *IEEE Access*, 2022.

[174] F. Dewangan, A. Y. Abdelaziz, and M. Biswal, "Load forecasting models in smart grid using smart meter information: A review," *Energies*, vol. 16, no. 3, p. 1404, 2023.

[175] X. Xia, J. Lin, Q. Jia, X. Wang, C. Ma, J. Cui, and W. Liang, "Etd-convlstm: A deep learning approach for electricity theft detection in smart grids," *IEEE Transactions on Information Forensics and Security*, 2023.

[176] R. Cárdenas, P. Arroba, J. L. Risco-Martín, and J. M. Moya, "Modeling and simulation of smart grid-aware edge computing federations," *Cluster Computing*, vol. 26, no. 1, pp. 719–743, 2023.

[177] F. Samie, L. Bauer, and J. Henkel, "Edge computing for smart grid: An overview on architectures and solutions," *IoT for Smart Grids: Design Challenges and Paradigms*, pp. 21–42, 2019.

[178] S. Shannigrahi, S. Mastorakis, and F. R. Ortega, "Next-generation networking and edge computing for mixed reality real-time interactive systems," in *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2020, pp. 1–6.

[179] K. Toczé, J. Lindqvist, and S. Nadjm-Tehrani, "Performance study of mixed reality for edge computing," in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, 2019, pp. 285–294.

[180] X. Kong, Y. Wu, H. Wang, and F. Xia, "Edge computing for internet of everything: A survey," *IEEE Internet of Things Journal*, vol. 9, no. 23, pp. 23 472–23 485, 2022.

[181] P. McEnroe, S. Wang, and M. Liyanage, "A survey on the convergence of edge computing and ai for uavs: Opportunities and challenges," *IEEE Internet of Things Journal*, vol. 9, no. 17, pp. 15 435–15 459, 2022.

[182] H. Hua, Y. Li, T. Wang, N. Dong, W. Li, and J. Cao, "Edge computing with artificial intelligence: A machine learning perspective," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[183] P. Barba, J. Stramiello, E. K. Funk, F. Richter, M. C. Yip, and R. K. Orosco, "Remote telesurgery in humans: a systematic review," *Surgical endoscopy*, vol. 36, no. 5, pp. 2771–2777, 2022.

[184] L. Tan, W. Su, W. Zhang, J. Lv, Z. Zhang, J. Miao, X. Liu, and N. Li, "In-band network telemetry: A survey," *Computer Networks*, vol. 186, p. 107763, 2021.

[185] D. Bhamare, A. Kassler, J. Vestin, M. A. Khoshkholghi, J. Taheri, T. Mahmoodi, P. Öhlén, and C. Curescu, "Intopt: In-band network telemetry optimiza-

tion framework to monitor network slices using p4," *Computer Networks*, vol. 216, p. 109214, 2022.

[186] S. Sheng, Q. Huang, and P. P. Lee, "A general delta-based in-band network telemetry framework with extremely low bandwidth overhead," *Computer Networks*, vol. 223, p. 109573, 2023.

[187] A. Sacco, A. Angi, F. Esposito, and G. Marchetto, "Hint: Supporting congestion control decisions with p4-driven in-band network telemetry," in *2023 IEEE 24th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 2023, pp. 83–88.

[188] J. Hyun, N. Van Tu, J.-H. Yoo, and J. W.-K. Hong, "Real-time and fine-grained network monitoring using in-band network telemetry," *International Journal of Network Management*, vol. 29, no. 6, p. e2080, 2019.

[189] P. Sun, J. Lan, J. Li, Z. Guo, and Y. Hu, "Combining deep reinforcement learning with graph neural networks for optimal vnf placement," *IEEE Communications Letters*, vol. 25, no. 1, pp. 176–180, 2020.

[190] S. Deng, M. Li, Q. Guo, and H. Zhou, "Security sfc path selection using deep reinforcement learning," in *International Symposium on Mobile Internet Security*. Springer, 2022, pp. 97–107.

[191] M. A. Khoshkholghi and T. Mahmoodi, "Edge intelligence for service function chain deployment in nfv-enabled networks," *Computer Networks*, vol. 219, p. 109451, 2022.

[192] M. Bunyakitanon, X. Vasilakos, R. Nejabati, and D. Simeonidou, "End-to-end performance-based autonomous vnf placement with adopted reinforcement learning," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 2, pp. 534–547, 2020.

[193] Y. Yue, X. Tang, W. Yang, C. Cao, and Z. Zhang, "A deep learning-based vnf placement approach for sfc requests in mec-nfv enabled networks," in *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, 2023, pp. 1–3.

[194] Delloite, "The future of health," *Last Accessed: October*, 2023. [Online]. Available: https://www2.deloitte.com/us/en/pages/life-sciences-and-health-care/articles/future-of-health.html