Context-based Security Function Orchestration for the Network Edge

Mircea-Mihai Iordache-Şică

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

April 22, 2022

Abstract

Over the last few years the number of interconnected devices has increased dramatically, generating zettabytes of traffic each year. In order to cater to the requirements of end-users, operators have deployed network services to enhance their infrastructure. Nowadays, telecommunications service providers are making use of virtualised, flexible, and cost-effective network-wide services, under what is known as Network Function Virtualisation (NFV). Future network and application requirements necessitate services to be delivered at the edge of the network, in close proximity to end-users, which has the potential to reduce end-to-end latency and minimise the utilisation of the core infrastructure while providing flexible allocation of resources. One class of functionality that NFV facilitates is the rapid deployment of network security services. However, the urgency for assuring connectivity to an ever increasing number of devices as well as their resource-constrained nature, has led to neglecting security principles and best practices. These low-cost devices are often exploited for malicious purposes in targeting the network infrastructure, with recent volumetric Distributed Denial of Service (DDoS) attacks often surpassing 1 terabyte per second of network traffic.

The work presented in this thesis aims to identify the unique requirements of security modules implemented as Virtual Network Functions (VNFs), and the associated challenges in providing management and orchestration of complex chains consisting of multiple VNFs The work presented here focuses on deployment, placement, and lifecycle management of microservice-based security VNFs in resource-constrained environments using contextual information on device behaviour. Furthermore, the thesis presents a formulation of the latency-optimal placement of service chains at the network edge, provides an optimal solution using Integer Linear Programming, and an associated near-optimal heuristic solution that is able to solve larger-size problems in reduced time, which can be used in conjunction with context-based security paradigms.

The results of this work demonstrate that lightweight security VNFs can be tailored for, and hosted on, a variety of devices, including commodity resource-constrained systems found in edge networks. Furthermore, using a context-based implementation of the management and orchestration of lightweight services enables the deployment of real-world complex security service chains tailored towards the user's performance demands from the network. Finally, the results of this work show that on-path placement of service chains reduces the end-to-end latency and minimise the number of service-level agreement violations, therefore enabling secure use of latency-critical networks.

Acknowledgements

This thesis is built on years of discussion, work, collaboration and support with incredible people. I would like to express my gratitude for their patience and guidance along this long and arduous process.

First, I would like to thank my thesis supervisor, Prof. Dimitrios P. Pezaros, who provided continued guidance, motivation, funding, and insightful discussions.

I would also like to thank Dr. Christos Anagnostopoulos for his continued support on the theoretical aspects of my work.

To the NetLab research students within the University, Dr. Simon Jouet, Dr. Richard Cziva, Dr. Levente Csikor, Kyle Simpson, Stefanos Sagriotis, thank you for friendship, collaboration, feedback, and the occasional pint.

Finally, I would like to thank my family for their unlimited support in undertaking this task. To my parents, Antoaneta and Ovidiu-Liviu: you have always shown unconditional understanding and this work would never be completed without you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Telecommunication Service Provider (TSP) infrastructures have been growing constantly to cater to subscriber demands since the early 2000s. The services offered, which are now ubiquitous, include Voice over IP (VoIP), Video on Demand (VoD), and modern online services and platforms. The increased demand is driven by the increase of connected end-users and widespread availability of new mobile devices (e.g., smartphones, wearables, tablets, sensors, and more). However, the increase in network sizes causes a phenomenal increase in operational cost for service providers. Today's service providers experience poor infrastructure utilisation, tight coupling with hardware services, and poor infrastructure control interfaces which fail to adapt to the requirements of emerging mobile applications and services. To combat the loss of revenue and subscribers, research has been motivated towards different aspects of infrastructure management, such as resource management, energy efficiency, networking, and security.

To cater to the requirements and expectations of end-users, operators have been using network services to enhance their infrastructure. Such services (e.g., firewalls, intrusion detectors, caches, proxies, load balancers, WAN accelerators, etc.) have been deployed as specialised hardware appliances physically hardwired into the network infrastructure. Surveys show that the number of hardware appliances (also called middleboxes) is at least equal to the number of routers for all network sizes, the capital expenses incurred from the acquisition of middleboxes for enterprise networks (between 10,000 and 100,000 hosts) reached $1m every 5 years [130]. To save on the capital and operational expenses, providers have begun adopting virtualised network services. This transition, referred to as Network Function Virtualisation (NFV), changes

how network operators design their infrastructure to decouple network functionality from physical locations. Since its emergence in 2012, NFV has gained significant attention from providers, resulting in many deployments in Data Centre environments.

The increase in the number of devices has directly led to a considerable surge in traffic volume traversing TSP networks. Operators rely on infrastructure network security services to ensure resilience, which are considered as critical, but require specialised hardware to operate in large networks. For example, many Intrusion Detection and Prevention Systems (IDPS) are unable to function in real-time, with modern deployments detecting undesired network behaviour hours after an event has occurred. Due to the complexity of the services, operators have difficulty deploying scalable security services to ensure resilience, while minimising the impact on end-users [45].

At the same time, a new concept called Multi-Access Edge Computing (MEC) or fog computing, has emerged to better support the presence of mobile network-connected devices. These concepts present an IT service environment with cloud computing capabilities at the edge of the home, enterprise, IoT, or mobile network, within close proximity to the end users [95]. Utilising the edge network inherently provides low-latency connectivity to services, allowing operators to offload the utilisation of their core network. Use of NFV at the network edge has, to the best of our knowledge, focused on delivery of user-oriented services (e.g., caches, video transcoders), while operators maintain functionality related to network resilience within their core infrastructure. Considerations for the distinct requirements and constraints related to the use of security services at the network edge are not taken into account.

This dissertation investigates how the newly emerging technologies can be applied to cyber-security functionality, with the objective of minimising impact on end users' Quality of Experience with minimal trade-offs to security best practices employed by network operators. It proposes the design and implementation of an NFV platform tailored towards rapid deployment, reconfiguration, and high availability of security services in distributed, heterogeneous, and resource-constrained networks. Building on top of the Glasgow Network Function principles [34], it presents the considerations for chaining multiple Virtualised Network Functions (VNFs) to provide complex network services. As network security is paramount to preventing and mitigating increasingly frequent attacks, deployment of such services in resource-constrained environments (e.g., 5G, IoT, Autonomous Vehicle Networks) is crucial for network operators. A series of lightweight, microservice-based security network functions are presented that can be composed into complex services.

Furthermore, this thesis advocates on-path deployment to reduce resource consumption and la-

tency overhead, with a non-sharing strategy for enhanced management and reduced complexity. Combined with SDN and NFV, this work enables service operators to provide resource-efficient, on-demand customised security services for end users by chaining and deploying network functions close to end users. Moreover, this work provides an optimal solution for the placement of security services problem based on Integer Linear Programming, and a heuristic solution targeting real-time allocation capabilities, saving the infrastructure's computing and communication resources.

1.2 Thesis Statement

The deployment of VNFs introduces flexibility and dynamism in response to the increased demand for network-enhanced services in modern and next-generation networks. Network operators' enforcement of cyber-security policies needs to respond to the temporal variations within the network, while providing service-level agreements to end-users. This work asserts that creating, dynamically managing, and monitoring lightweight security modules in edge networks (for example public wireless LANs, 5G cell clusters, Autonomous Vehicle networks) will allow operators to provide assurances on device-to-device communication. The work focuses on network functions that, through behaviour and placement within the network, lead to low traffic latency overhead in edge network paths. The flexible nature of security best practices requires that the work not limit the expression of security service composition and placement.

1.3 Contributions

The thesis contributes to the development, management and orchestration, and placement necessary for security functions to operate in edge networks through:

- A comprehensive review of past and current approaches that aim to introduce programmability within the network.
- An in-depth analysis on the current network security functions, their operational constraints, and resulting limitations in dynamic placement, management and orchestration when applied on edge networks.
- The design of management mechanisms required for operation in resource-constrained environments, in agreement with established NFV design standards.

- A formulation for latency-optimal placement of on-path Network Service Chains problem and an exact solution using Integer Linear Programming and the Gurobi solver.
- The implementation of management and orchestration modules tailored for use in edge networks, where the availability of high-performance middleboxes is scarce.
- Proof-of-concept implementation of common security network functions in a lightweight, composable fashion to minimise packet processing overhead.
- The design and implementation of a heuristic, Minimal Path Deviation Allocator, to solve the placement problem in highly dynamic, roaming user environments.
- An evaluation of the benefits of lightweight security network functions compared to monolithic architectures and traditional hardware appliances.
- An evaluation of the performance benefits of dynamically creating and managing security network functions in roaming user environments.
- An evaluation of the heuristic placement algorithm over a nation-wide, simulated network topology consisting of edge and cloud servers.

1.4 Organisation of the Thesis

The work presented in this thesis is structured as follows:

Chapter 2 discusses the need and evolution of virtualised, software-based network infrastructures and introduces the emerging requirements of today's network users. It outlines the concepts of Network Function Virtualisation (NFV) and Software Defined Networking (SDN). It then describes the challenges that large-scale telecommunication networks face and the incentive for adoption of the newly available edge architecture alongside the existing infrastructure. This chapter presents the challenges in assuring infrastructure resilience through security services and the evolution of these solutions. It depicts the limitations of current approaches and shows how device-centric paradigms can be integrated for enhancing networks. The chapter discusses current solutions for network-wide deployment of security services and existing management solutions for them.

Chapter 3 critically discusses the limitations of previous network security paradigms and orchestration algorithms, and motivates the need for a modular, device-centric approach. After introducing the most important design considerations, this chapter compares different security

architectures (e.g., microservices, monoliths, and hardware-accelerated) and their suitability for the network edge. Furthermore this chapter identifies the challenges for a responsive orchestration mechanism and presents a latency-optimal in-line service chain placement algorithm.

Chapter 4 details the technical aspects of the implementation for the design presented in Chapter 3. A bottom-up framework is presented, detailing modular security service implementations, their placement and lifecycle management strategies, and the network traffic in several deployment scenarios. It considers real-world constraints in providing allocation of complex service chains, and proposes a heuristic approach that conforms to the dynamic behaviour of roaming clients.

Chapter 5 provides a comprehensive evaluation of the proposed framework. First, it shows some characteristics of modular security services that are important for operational reliability (e.g. overhead, instantiation time, delay, throughput). It then presents how the lifecycle management performs in environments with roaming clients (e.g., Autonomous Vehicle networks, Sensor Networks). Finally, this chapter presents an evaluation of the latency-optimal in-line service chain placement orchestration and its heuristic approximation over a simulated network topology with real-world latency characteristics.

Chapter 6 summarises the work and impact of this thesis. It also presents future research directions on the topic.

Chapter 2

Background and Related Work

2.1 Overview

The ARPANET, the foundation upon which the Internet is constructed, was designed 50 years ago as a way for a limited number of researchers to transmit electronic messages and scientific results [120]. Since its inception, the Internet has seen a tremendous increase in the number of users and traffic traversing it. This has prompted network operators to investigate strategies for how to manage network infrastructures and provide increased resilience [122]. However, the management of such widespread infrastructure in the face of malicious intent (e.g., interfering with normal operation, bypassing security and privacy mechanisms, or limiting expansion capabilities by oversubscribing resources) is an ongoing challenge [9] [49] [159].

In order to facilitate efficient resilience mechanisms and their management and orchestration, both academia and industry have made considerable efforts since the early 1990s, when the Internet and its diverse applications had started becoming widespread among general users. Contributions have been focused around enhancement of network programmability, with early ideas including the Active Networking paradigm [145] [136]. The recent widespread adoption of general-purpose commodity and resource-constrained hardware located at the edge of the network [90] (e.g., Internet of Things, Smart Sensors, Fog Computing, etc.), coupled with the Network Function Virtualisation concept [91], have enabled the use of advanced, programmable network services in close proximity to the users with minimal modification of the underlying network infrastructure [36]. Recent advances have enabled a shift from legacy networking, where services are tightly coupled with their network location (and, by extension, difficult to expand upon), to a delivery model for user and location-specific network services.

In this chapter, the background and related work of this thesis are outlined. First, in Section 2.2 research leading to network programmability and virtualisation through SDN and NFV is presented starting from research beginning in 1990s. Then, in Section 2.3, an overview of how moving network services closer to the user, at the network edge, is achieved and highlights some of the benefits of the approach. Particular attention is given to orchestration challenges in Section 2.5 with details on next-generation, edge-based services. Section 2.4 describes the advances made in the management and orchestration of security services for enhanced network resilience, with a focus on how paradigms have shifted with the growth of the Internet. Finally, Section 2.6 summarises the key findings of this chapter.

2.2 Virtualisation of Network Infrastructure using SDN and NFV

In this section, the work leading up to network programmability through Software Defined Networking (SDN) and Network Function Virtualisation (NFV) is reviewed. A timeline, starting from the 1990s, when the Internet started gaining popularity, is established. This review presents highlighted works (e.g., active networking, ForCES, Ethane) and how they paved the way towards current paradigms of SDN and NFV, the state-of-the-art mechanisms for delivery and management of softwarised network services in an automated manner.

2.2.1 Towards network programmability

In the 1990s, many new applications beyond the transfer of messages and data between scientists have started making use of the Internet [46]. Many such applications have been built on top of existing network protocols, but the improvement and refinement of network services and their behaviour became a significant challenge. This is due to the multi-vendor, distributed nature of networks and slow protocol adoption coordinated by standards bodies (e.g. IETF).

A clean-slate approach was proposed in the 1990s [135] through *Active Networking*, that aimed to evolve networks at a large scale. This paradigm aimed at exposing through an application programming interface the storage, processing resources, and control of packet queues of various network devices [21]. Using the given API enabled the dynamic alteration of the behaviour of individual network nodes to implement specific network functions for a subset of packets passing through the node. The operations proposed were carried in-band, encapsulated within

data packets [145]. The envisioned applications for this new approach included fine-grained control over packet forwarding mechanisms and dynamic network support for new applications. The evolution of *Active Networking* offered the vision of unified middlebox control instead of ad-hoc approaches [136].

Disagreement within the research community on the behaviour of the network, with no consensus reached between simple packet forwarding or processing for enhanced functionality, proved to be a major challenge in adoption of the technology. Privacy and safety concerns were raised on the ability to run executable code within the network, with these aspects mostly overlooked by the wider research community. The proposed flexibility through the encapsulated applications brought into question the performance of the entire system, leading to further challenges in adoption. Finally, the lack of viable commercial deployments, as there was no need for network programmability at the time, restricted the adoption beyond research projects [46].

2.2.2 Separation of Control and Data Planes

In the early 2000s, with a mostly established Internet infrastructure, the demand for higher performance and increased reliability shifted the primary focus of service providers. With the number of devices connected to the Internet passing 1 billion in 2005[1], the need for network management functions, known as Traffic Engineering (TE), became apparent. Network operators and researchers started investigating methods for controlling the physical path used to deliver traffic [141]. Initially, this was only possible through conventional, but primitive, routing protocols which restricted and tied into conventional routers and switches, leading to complex and extremely difficult network management tasks (e.g., customised routing, telemetry, and debugging).

The continual improvement in speed of commodity and general purpose computers, with new processors and high speed memory being released yearly, quickly outpaced the control planes of networks [46]. These limitations led to two major innovations in the field: *open interfaces* between the control and data planes, and *logical centralisation of the control plane*.

The resulting ForCES (Forwarding and Control Element Separation) working group at the IETF proposed a standardised interface between control and data planes [152]. Logical centralisation of control was proposed using protocols such as the Routing Control Platform [19]. These concepts, while novel within IP networks, have been already known in the other industries for some time. For telecommunications, SS7 proposed signalling plane separation in the early 1990s [99].

[1] https://ourworldindata.org/internet Retrieved April 2020

The concepts saw little adoption, as it exposed internal properties of networks and dissuaded vendors from providing implementations. In order to appease hardware providers and provide an incentive for widespread adoption, SANE and Ethane proposed a logically centralised flow-level approach for access control [23] [22]. Ethane abstracted the switch into a set of flow tables, with entries populated by a high-level controller [22]. Over time, with industry support and a well-defined control protocol, Ethane evolved into OpenFlow [94], the project which shaped the modern networking industry.

2.2.3 Improvement of the Control Plane

Success of experimental infrastructures in the mid-to-late 2000s (e.g., [26]) led to increased interest in large scale network experimentation. Example research projects included the European funded Future Internet Research and Experimentation (FIRE) initiative [51] and its US counterpart, Global Environment for Network Innovations (GENI) [15], which investigated how networks could be managed at scale. These projects pushed forward the concept of Software Defined Networking (SDN), where the desire was to provide programmability into the network for better infrastructure management and operation. The networking community has been split between the desire for highly-programmable networks and the realistic limitations of current infrastructure equipment that is costly to replace.

2.2.4 The OpenFlow Protocol

OpenFlow [94] was proposed by researchers at Stanford as an incentive that aimed to balance programmability and pragmatism. The switch design, inherited from Ethane [22], leverages existing packet-processing hardware present in commodity switches. The programmability aspect, compared to earlier proposals, is greatly enhanced. Because of the design decision to achieve balance between the two major factors mentioned above, OpenFlow saw a rapid rise in adoption within both research and industry.

The principle of operation is straightforward, and brings innovation in the network control plane. It defines clear separation of the network control plane, which specifies routing policies and route finding algorithms, from the data plane hardware that forwards packets, as seen in Figure 2.1. Furthermore, the control plane is logically centralised within a well-defined SDN controller that maintains a global view of the network and all associated data plane elements. This logical centralisation greatly facilitates the deployment of new control procedures (e.g. routing, engi-

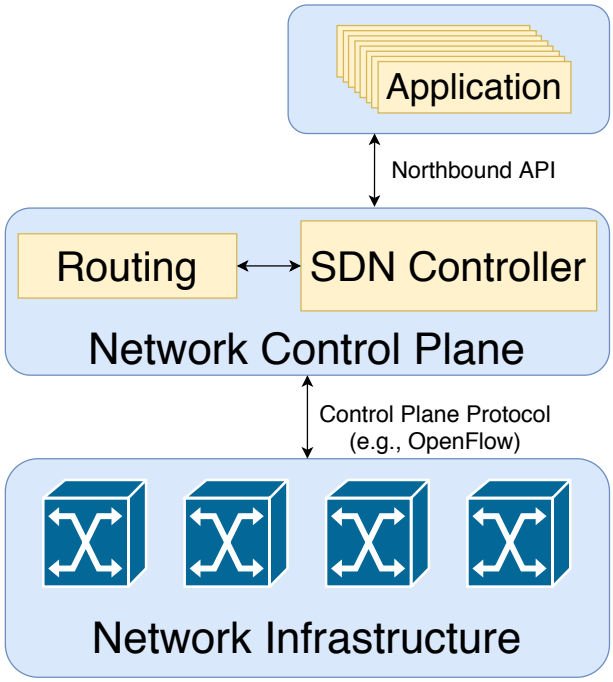

Figure 2.1: An overview of the Software-Defined Networking architecture

neering, services) in a single place [94] [10] [4], instead of individual modification of all network devices involved. The shift towards SDN allows the network to be treated as 'one big switch'.

In implementing SDN, the OpenFlow specification [61] defines a data forwarding pipeline along with a TCP-based communication protocol that is used between the SDN Controller and the data plane devices located within the network.

The data forwarding plane is based on multiple match-action tables that hold a fixed number of flow entries that describe a matching pattern and the associated actions (e.g., forwarding, dropping, modifying field headers). This is enhanced with a set of counters to track the number of bytes and packets matched, along with priority information that is used when a packet matches multiple entries.

The associated OpenFlow communication protocol, built on top of TCP, allows creation, removal and modification of flow entries, alongside queries related to flow statistics. While multiple revisions of the OpenFlow protocol have been developed since its inception, the following three can be considered prominent: OpenFlow 1.0 (the initial released, used even today), OpenFlow 1.3 (the de-facto standard in current hardware implementations), and OpenFlow 1.5 (the most up-to-date version which has started gaining popularity at the time of writing).

From a service provider's perspective, the short timescales required for application of new policies, coupled with the highly scalable control plane, permitted the rapid adoption of the tech-

nology. Research has made use of the resulting network programmability to implement a range of functions such as Quality of Service enforcement [10], network virtualisation [4], flow-based routing [4], to management of Virtual Machine placement in cloud Data Centres [33].

2.2.4.1 SDN Deployments, Switches and Controllers

The introduction of SDN concepts was rapidly adopted by various large-scale network and data centre operators. One of the first deployments was reported by Google in 2013 [70], being used to interconnect private DC's at a global scale. Known as B4, it allows setting up bandwidth guarantees between any two hosts regardless of their location. The paper describes how multiple routing protocols are supported simultaneously and how centralised traffic engineering is performed.

Many switch vendors have provided hardware implementations that align with the SDN vision. Well-established equipment vendors such as Cisco and Juniper Networks added OpenFlow capabilities to a range of devices alongside their own control protocols. Examples include the Cisco Nexus 9000 series[2] or Juniper Networks MX series edge routers[3]. New manufacturers, such as Barefoot Networks and Noviflow, have started manufacturing devices for the sole purpose of SDN, with high-speed programmable pipelines and more recent versions of the OpenFlow protocol. Such an example is NoviFlow's NoviSwitch 21100[4], which offers up to 512Gbps and 360Mpps switching capacity, support for all OpenFlow 1.3 and OpenFlow 1.4 match fields, as well as key OpenFlow 1.5 features. Alongside hardware solutions, software switches have been designed for SDN, with notable examples being Open vSwitch [109], Pisces [129], and mSwitch [63].

Network operators benefit from a wide variety of SDN Controllers. Popular alternatives include OpenDaylight [96], ONOS [14], and NOXPOX [54], all of which are used by many enterprise network providers (e.g., Huawei, AT&T, Vodafone). Research projects have also made use of the Ryu [5] controller for its lightweight nature, ease of development and support for the latest versions of the OpenFlow protocol.

[2]https://www.cisco.com/c/en_uk/products/switches/nexus-9000-series-switches/index.html Retrieved April 2020

[3]https://www.juniper.net/documentation/en_US/release-independent/junos/topics/reference/general/junos-sdn-openflow-supported-platforms.html Retrieved April 2020

[4]https://noviflow.com/wp-content/uploads/2019/11/NoviSwitch-21100-Datasheet-400_V5.pdf Retrieved April 2020

[5]http://osrg.github.io/ryu/ Retrieved November 2020

2.2.4.2 Beyond OpenFlow

SDN offers a vision of centralised, virtualised control plane working alongside a lightweight data plane. The technology allows centralisation of previously-distributed control applications, such as access control management, network topology management. It allows separation-of-concerns, delimiting the control plane (with well-defined messages) from the data plane.

On the other hand, SDN design is outside the scope of data-intensive operations, with the controller incurring significant overheads and delays on the network in these scenarios. If every packet within the network were to be redirected to the controller for inspection, the benefits of a logically centralised controller would be overshadowed by the resulting performance degradation.

This limitation has inspired new research directions in the field of programmable data planes. Researchers started to investigate how data planes could be defined in a programmatic fashion. Examples include P4 [16], which aims to define switch pipelines, and BPFabric [74], which uses a platform-independent instruction set for data-plane functions.

Another approach is to use Network Function Virtualisation to perform data-plane processing. Details of this paradigm are presented in the next section.

2.2.5 Benefits of the Programmable Control Plane - Network Function Virtualisation

The concept of Network Function Virtualisation (NFV) is used to extend network service availability. The paradigm proposed by NFV is to replace hardware-based specialised appliances, that implement high-level packet-processing functionality within the data plane, with virtualised software artifacts. The resulting Virtual Network Functions (VNFs) are able to run on Off-The-Shelf hardware by using widely available programming languages, frameworks and virtualisation concepts [98].

Decoupling the network functions from the underlying hardware enables faster development, deployment, and provisioning of network service functions. As a result, network services can be decomposed into multiple VNFs running on physical or virtual machines and addresses the compatibility with vendor-specific hardware and control mechanisms [11].

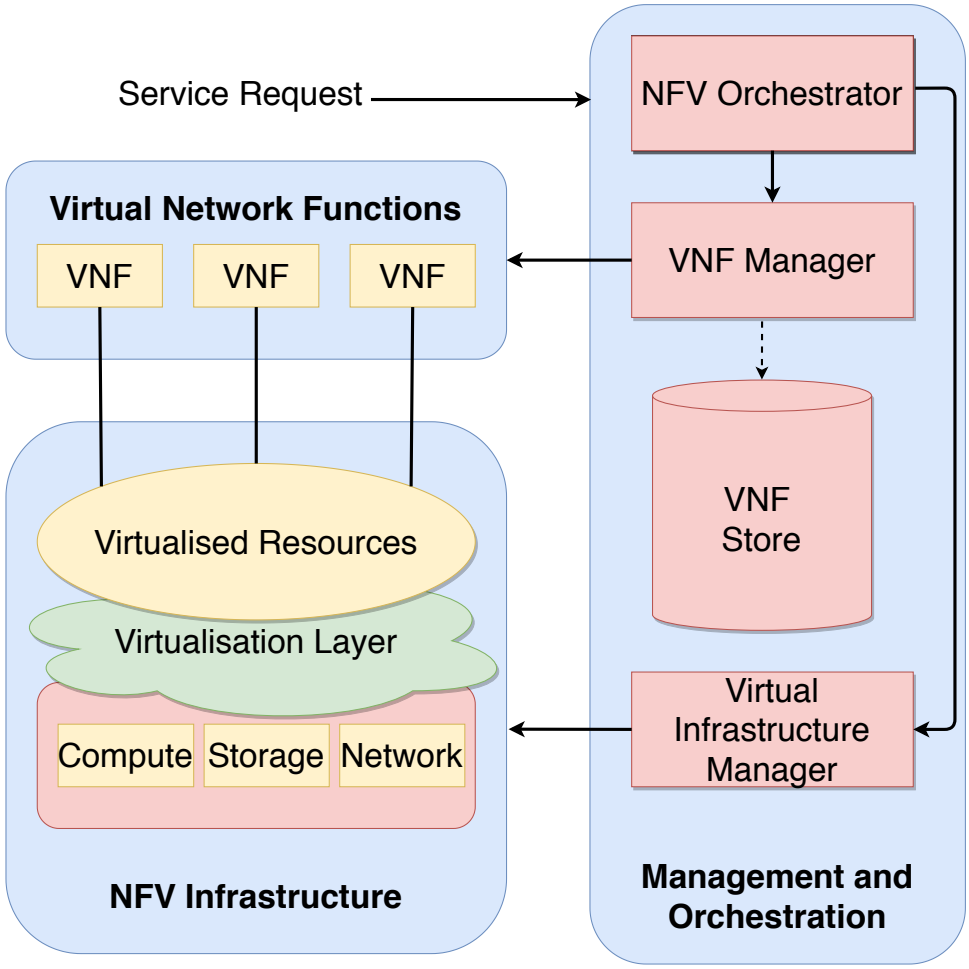

Figure 2.2: The proposed ESTI NFV Architecture

2.2.5.1 NFV Reference Architecture

To accelerate adoption of NFV and help in development of coherent and inter-operable VNF implementations, the European Telecommunications Standards Institute (ETSI), the standardisation body for NFV, has put forward a reference architecture of an NFV platform [42]. Since the first release in 2012, the proposed architecture has been the basis for most NFV frameworks. At a high-level, the architecture is composed of three distinct components: the Management and Orchestration block (MANO), the NFV Infrastructure (NFVI) and the Virtual Network Functions themselves, with Figure 2.2 illustrating how they interact.

Management and Orchestration (MANO): The MANO component is responsible for overall control of the NFV system [116]. It receives as input a VNF Service Request, and outputs instructions for the VNF Infrastructure for provisioning and allocation, as well as configuration messages to the instantiated VNFs.

The NFV Orchestrator (NFVO) has high-level control over the NFV process, parsing the Service Requests and determining the respective instructions for other sub-modules. According to the orchestration strategy used, it determines allocation of the requested VNFs, reacts to events impacting performance, and monitors the overall health of the system. It communicates directly with the VIM and VNFM components.

The VNF Manager (VNFM) is responsible for setting up the software implementing the VNF functionality within the virtualised resource. It typically links up with a database or VNF Catalogue that describes how particular VNFs are to be configured.

Finally, the Virtual Infrastructure Management (VIM) component communicates with the physical infrastructure to determine resource availability and handle creation and destruction of virtual resources to be used by VNFs. It is also responsible for performing the health check of the physical infrastructure.

NFV Infrastructure (NFVI): The NFV Infrastructure (NFVI) block of the ETSI Architecture consists of the physical resources employed in the operation of VNFs. It abstracts the hardware resources available (e.g., compute, storage, IO, network availability, etc.) through virtualisation (e.g. x86 virtualisation using Virtual Machines processor support). The result is a logical segmentation of resources for use by multiple VNFs.

The NFVI component communicates directly with VNFs, controlled by the virtual infrastructure manager (VIM) of the MANO module. NFVI enables the use of commodity hardware for multiple network services, a critical component in achieving the economic benefits that NFV proposes.

Virtual Network Functions (VNFs): The Virtual Network Functions (VNFs or NFs), are the software artifacts that implement network services. They are encapsulated in a virtualisation framework (e.g. Xen, Docker, Linux Containers, etc) and can be deployed on a specific server in the infrastructure. Similar to other virtualisation technologies it enables migration between servers and deployment and shutdown in short time frames (seconds or minutes) [91] [36].

The implementation of VNFs can be done using virtual compute, storage IO and network resources. Management is done by the VNFM component of the system.

2.2.5.2 Relationship between SDN and NFV

The ETSI NFV Specification and Architecture does not include SDN, but the two technologies are often used in tandem. Many different ways of enabling integration have been proposed.

In general, NFV provides the desired programmability to application layers of the network stack, while SDN enables control over the lower layers of the network. For example, NFV allows the rapid creation of a firewall VNF allowing only web traffic, while SDN would be used to automatically re-route traffic to this new service.

In achieving this synergy, researchers identified and overcame several challenges. Foremost is the transparent routing of network traffic through VNFs. In 2013, Zhang et al. proposed StEERING [155], an SDN approach for routing traffic destined for network services. The paper led to a plethora of related research of 'flow steering', the technique which redirects unidirectional network flows. This work evolved with specialised applications for Optical Networks [150], Multi-Tenant Data-Centre Networks [139], and Cloud-based Edge Networks [36].

Another challenge related to the synergy between SDN and NFV is the dynamic encapsulation of packets for transparent routing and processing of packets for traversal of multiple VNFs. FlowTags is one of the first initiatives to attempt this [45], by using IPv4 headers to encode information regarding VNF processing. OpenSCaaS [45] investigates the use of existing tags and fields, or the use of a new network header and implications on network switches, middleboxes, as well as impact on scalability and conflicts with existing network services (e.g., VLAN IDs). The IETF has proposed the Network Service Header (NSH) [115], a new field inserted into packets that can be used to realise service function paths. Use of NSH allows the exchange of metadata between different VNFs.

2.2.6 NFV Packet Capture and Processing

A key component that enabled the rapid research, development, and adoption of Network Function Virtualisation as an essential paradigm is the development of rapid, yet portable, packet processing frameworks. By definition, NFV requires I/O-intensive operations, as opposed to the traditional compute-bound tasks, which affects the design of memory access, cache locality, and synchronisation primitives utilised.

To facilitate development of network I/O-centric applications, such as VNFs, several frame-

works have been proposed. Although the scope of the thesis is to provide a general-purpose solution, its implementation is inspired by, or benefits from, some of the notable works presented below.

Click: The Click Modular Router proposal [78] was first introduced in 2000 as a new software architecture for building flexible and configurable routers. Aiming at the development and deployment of routers to experiment with new networking technologies, Click proposes a combination of individual packet-processing modules, called *elements*, that implement simple router functions. Behaviour of the Click router is defined by a directed graph, with *element* vertices, and packet flow described as graph edges. The authors propose an IP Router implementation that comprises of 16 elements and evaluate the solution's performance on commodity hardware. The popularity of Click has been expanded with ClickOS [91], a lightweight, Xen-compatible[6] implementation that is able to deploy Click dataplane processing middleboxes. ClickOS has proven to be able to run multiple Click middleboxes on commodity hardware, and being able to achieve high throughput.

netmap: Rizzo [119] proposed netmap as an alternative for user-space applications to gain fast network packet receive and transmit capabilities. In order to achieve this result netmap uses pre-allocated packet buffers (which eliminate per-packet memory management), zero-copy mechanisms using protected access to said buffers, and batching of system calls. Furthermore, data structures which replicate hardware features (such as *netmap rings*, which behave as a circular rings for Tx and Rx) allow for better utilisation of the underlying hardware.

extended Berkeley Packet Filter (eBPF): BPF [93] is a technology that allows for rapid access to raw network interfaces, and allows analysis and modification of network traffic. It is widely available on most Unix-like systems, with Linux and Microsoft Windows, providing an extended implementation which supports Just-In-Time (JIT) compilation of eBPF programs. The programming model of BPF is oriented towards traffic matching and filtering, being represented as a high-level description of intent. Extensions to GCC and LLVM compilers allow for compilation of a subset of C code into eBPF bytecode. Because of wide adoption, the paradigm is encountered in notable works, e.g., Jouet et al. [74] proposed a programmable switch building on eBPF technology.

eXpress DataPath (XDP): XDP feature[7], part of the Linux kernel as of version 4.8, allows for deployment of eBPF programs early within the kernel packet processing path, before memory

[6] Xen is a popular virtualisation hypervisor within Data Centre environments. https://xenproject.org Retrieved September 2021

[7] https://www.redhat.com/en/blog/capturing-network-traffic-express-data-path-xdp-environment Retrieved September 2021

allocation occurs. The eBPF programs supplied have verification checks to ensure there are no loops, no global variables, and no out-of-bounds accesses. Programs within XDP are allowed to modify the packet and output packet actions, such as dropping the packet, ingressing into the network stack, or redirecting to a NIC. The main limitation of XDP is the associated support provided by the NIC driver; a fallback driver is available which performs processing within the network stack with lower performance.

Data Plane Development Kit (DPDK): Originally released by Intel in 2010, DPDK [48] offers a set of primitives that help creating efficient I/O-oriented NFs. These primitives include memory management (e.g., *rte_malloc*, or *rte_mempool*), and IPC (*rte_ring*) primitives, and allow for cache-local, NUMA-socket allocation of resources. The recommended development model requires each process to be allocated one full CPU core, and the number of concurrent processes is limited by the hardware architecture. Multi-process applications are supported through IPC and resource sharing (*rte_ring* and *rte_mempool*, respectively). Finally, DPDK requires the use of a *PollMode Driver* library to interface with the physical Network Interface Cards (NIC) without the overhead of an operating system networking stack.

2.2.7 Implementations of NFV Architectures

Many NFV platforms have been proposed, both in academia and industry. The frameworks put forward are usually targeted towards specific network environments (e.g., ISP Core infrastructures, Data Centre networks, telecommunications networks, etc). The following paragraphs provide details on prominent NFV frameworks, with some of their unique aspects.

OSM: Open Source MANO (OSM)[8] is the ETSI-hosted project that aims to provide an Open Source NFV Management and Orchestration (MANO) software stack aligned with ETSI NFV. Many corporations, including Amazon, Intel, BT, Telefonica and T-Mobile back the initiative.

In order to facilitate widespread adoption of the NFV paradigm, and ensure that the above-mentioned specification is followed, OSM offers a VIM-independent, production-quality software stack that integrates with different management software. The design decisions are made to increase interoperability among multiple NFV implementations with a well-defined model. As a result, the proposed framework has rapidly seen deployment within core network infrastructures.

OpenStack: OpenStack[9] is designed to control and manage large pools of compute, storage,

[8] https://osm.etsi.org Retrieved May 2020
[9] https://www.openstack.org Retrieved May 2020

and network resources. Management is performed through a dashboard or the OpenStack API. The project is free and open-source, with over 500 companies contributing to its development. It has gained popularity over the past decade, being adopted by many datacentre operators and cloud providers (e.g., RackSpace, Tencent Cloud, Huawei, etc.).

The main advantage of OpenStack is the large-scale resource pools available for management, being able to control thousands of hosting servers running tens of thousands of Virtual Machines. As a result, many NFV platforms have been developed on top of OpenStack, but it is limited with respect to the orchestration component and integration with network controllers.

The VM placement algorithm used to determine physical location of VNFs is implemented in the *nova-scheduler* module. The algorithm involves host weighting (e.g., based on available resources on hosts) and filtering (removal of hosts that are not supposed to run a VM) pipelines before allocating the VM to the least-loaded host.

OpenMANO: OpenMANO [87] is an open-source project led by Spanish telecommunications operator Telefonica, with the goal of implementing the ETSI NFV MANO specifications. The implementation focuses on providing enhancements in performance and portability. The core OpenMANO framework consists of the *openmano*, *openvim*, and graphical interface.

openvim is a lightweight reference implementation of the NFV Virtual Infrastructure Manager. Some of the features include NUMA, CPU, memory affinities (assignment of specific hardware resources to VNFs). The main innovation is combining OpenStack integration for resource management with SDN Controller communication to perform the required network service configurations.

openmano is a reference implementation that offers creation and deletion of VNF templates and instances, as well as network service templates and instances. It interfaces with *openvim* through its API, and provides a northbound API for integration with Graphical User Interfaces.

Orchestration is done using one of three VM providers that OpenMANO integrates. If used with OpenStack, the default placement with *nova-scheduler* is performed, as described above. Integration with VMware's vCloud Director [24] is possible, using the provided proprietary placement algorithm that optimises for balanced load hosts and high availability. Finally, OpenMANO can be used with public cloud systems (e.g., Amazon AWS) where physical placement of VNFs can only be controlled at a high-level.

Kubernetes: Kubernetes[10] is an open-source production-grade container orchestrator designed to automate deployment, scaling, and management of containerised applications. Initial development was performed at Google, and is currently maintained by the Cloud Native Computing Foundation, a partnership between Google and the Linux Foundation. The system was designed to support multiple container frameworks, most notable of these being Docker.

The reference Kubernetes scheduler (the component responsible for container placement) takes available resources from hosting nodes into account. Additional hints can be provided for host affinity. As with other general-purpose orchestrators, there is little-to-no integration of network information (e.g., link-layer latencies, hop counts) to provide enhanced placement.

Design decisions that influenced Kubernetes development limit the existing networking capabilities required for effective NFV implementation. Support for multiple, distinct virtual networks and use of high-performance network IO features is currently scarce, with extensive modifications and workarounds required for use in NFV scenarios [27]. On the other hand, Kubernetes does provide core Management and Orchestration functionality [52] that enables large-scale NFV deployments.

Cloud4NFV: Cloud4NFV [111] is one of the earliest NFV platforms, with origins in academia. The work focuses specifically on data modelling that allows for VNF description (e.g., VNF images, VM instances, storage, ports and network io requirements) as well as virtual infrastructure availability (e.g., cores, memory, ports, available VMs). The authors provide an implementation that allows VNFs to be mapped to physical resources using the presented data model.

Cloud4NFV is built on top of OpenStack, by default using the placement algorithm provided by *nova-scheduler*, as described before.

GNF: The Glasgow Network Functions (GNF) [35] framework is one of the first research-oriented NFV-centric projects that aims to use containers for hosting VNFs. The work proposes the use of commodity off-the-shelf software components to provide VNF hosting functionality. The main components of the framework are the *GNF Router*, *GNF Manager*, and *GNF Agent*, with a graphical user interface also provided.

The *GNF Agent* provides the operational logic and provides the NFV Virtual Infrastructure Management functionality. It resides on the servers hosting network functions and provides the necessary functionality for retrieving, instantiating and running VNFs on hosts. Additionally, it implements host-level traffic routing functionality for management of service chains and

[10]https://kubernetes.io Retrieved May 2020

provides information on the temporal resources and status of hosts.

The *GNF Manager* module is used to perform MANO of VNFs within the network. It is responsible for global control of VNF lifecycles, and communicates traffic routing requirements to the *GNF Router*, with which it is collocated. The component provides a southbound connection to the *GNF Agent* to retrieve host information and delegate container allocation, and a northbound REST API for basic VNF control primitives (e.g., creation, deletion, starting, and stopping) and global network overview.

For orchestration, a latency-optimal VNF placement scheme is used to minimise end-to-end latency. Physical VNF placement takes into account host resource availability, individual VNF requirements, and network-level information. The orchestrator implements dynamic VNF migration based on temporal network-wide latency fluctuations using optimal stopping theory to improve critical network parameters (e.g., latency, bandwidth) [32].

2.2.8 Composition of VNFs

Creating complex Virtual Network Functions from smaller components has been proposed as a method of achieving flexibility in service deployment [82]. Separating the behavioural elements into three overarching primitives — **Reception**, **Processing**, **Transmission** — proposes minimising the repeated operations (e.g., packet parsing) that take place on a singular middlebox in order to improve performance.

One of the first proposals towards this goal took the form of SoftNIC [60], which provided an enhanced hardware abstraction layer between the physical network interface and multiple VNFs operating on the same middlebox. The framework proposes the separation of network-specific operations (e.g. packet reception, decoding, encoding, and transmission) from VNF functionality. As a result a high-level VNF can be composed from multiple sub-functions, each of which operating only with the required information for adequate functionality. The architecture requires a multi-core middlebox, which allocates the SoftNIC component to one processing core, and subsequent VNFs to the remaining unused cores. Furthermore, each VNF requires communication with the SoftNIC component both for receiving and sending the relevant data, which increases overall processing delays.

Laufer et al. [82] proposed the consolidation of multiple Click VNFs [78] into a singular instance in order to reduce system-level Input/Output (e.g., from memory transfers, or cross-core communication). Aimed at providing operator-driven optimisation of VNFs, it provides multi-

ple Application Programming Interfaces for enhancing the OS Network Stack (as the base Click implementation features limited high-level network processing abilities), and providing Blocking I/O Operations. The work is however limited by the explicit requirement for operators to implement and optimise the CliMB extensions for each of their VNFs, and no clear separation between the three primitive operations mentioned above, limiting the composition of VNFs in highly dynamic environments.

Building on these principles, SNF [76], and later Metron [75], propose the use of composition of high-level VNFs through a clear separation of the three primitives involved. Support for hosting multiple primitives onto the same processing core provides minimal overheads from hardware-related data transfer, and provisions for dynamic composition of VNFs are made through grouping of multiple VNF processing classes and using internal ring buffers to interconnect with blackbox VNFs (the behaviour of which cannot be modified).

2.3 The Network Edge

As outlined in Section 2.2, the introduction and adoption of network programmability and virtualisation has shaped the landscape of the networking field. Modern networks are now able to leverage SDN and NFV in order to programmatically control, manage and automate operation.

This section introduces the evolving infrastructure located in close proximity to end-users and investigates the advantages of provisioning services on these networks.

2.3.1 Benefits of the Network Edge

Distribution of intelligent services throughout the network is the cornerstone of next-generation network implementations. By 2025, a Huawei Report estimates that there will be more than 100 billion connected devices, and mobile network coverage will reach over 6.5 billion people, an estimated 80% of the global population [151]. The new devices include smart mobile phones, wearables (e.g., smart watches, fitness trackers), networked household electronics, large-scale sensor networks, actuators, unmanned vehicles, etc.

The growth of mobile and smart devices pose new challenges for network operators. The amount of data generated by these devices is predicted to increase exponentially, and networks will be required to efficiently transfer the information [154]. Operators are aiming to provide new,

high-performance on-demand services to their users to be considered viable. Example services include always-on connectivity, high-definition video streaming, and rapid content delivery [90].

To achieve the customised services that operators are targeting, distributed network-wide intelligence is proposed by Mahmud et al. [90]. By expanding the network infrastructure close to the end users (the Network Edge), operators can meet performance goals and minimise unnecessary utilisation of the core network. The key benefits are summarised below.

Low Latency: Latency for a service is the temporal delay incurred by propagation, transmission, queuing, and processing of information. Propagation delay is the time required for a packet to travel from the sender to its intended destination over the medium, and is influenced by the distance travelled. For general network edge environments, these distances range between a few meters (in the case of dense small-cell networks: e.g., 5G, WiFi [7]) to a few kilometres (for customer-provided equipment connected to a demarcation point). The short distance results in reduced propagation delays and can be leveraged when services are hosted on edge devices. Predictable low latency can be used to increase network reaction times in response to events, improve user experience, and enable delay-sensitive applications (e.g., augmented/virtual reality, unmanned vehicles) [107] [111] [128].

Proximity to Data Origin: Being near to the source of data is essential in minimising network congestion and reducing unneeded utilisation of upstream networks. Proximity to data sources is a key factor in capturing information intended for real-time processing or analytics. To illustrate this concept, Wang et al. [146] proposes a network edge-based architecture for real-time live video analytics and processing that significantly reduces bandwidth demand with minimal impact on accuracy.

Location Awareness: If devices are connected to wireless networks (e.g., 5G, WiFi) they can extrapolate low-level signalling information to accurately determine the location of other users. A prime use-case for this is in an unmanned vehicle environment where sensor information can be used in conjunction with wireless location to accurately determine the position of other equipment [133].

Contextual Information: Contextual information offered by edge devices can enhance user experience and enable monetisation. New applications that aid users in identifying products, provide health information, or help retailers in analytics can emerge [25].

On-Premise Protection: Edge devices can be located on-premises, and isolated from the public Internet. This is crucial with health-related sensitive data, as it has to be managed and processed

without being sent over the to cloud services [131].

2.3.2 Trends in Edge Computing Research

The topic of Edge Computing is currently being investigated from different perspectives. These trending research directions and topics have a unique convergence: an intermediate layer of computation and storage between the user and the cloud. This section expands on the most popular approaches in Edge Computing.

2.3.2.1 Multi-access Edge Computing

In one of the early and defining works regarding edge computing, the term cloudlets has been coined by Satyanarayanan et al [125]. Their work aims to bring cloud functionality to mobile users by leveraging computational resources located in close proximity to network users. By avoiding WAN delays, jitter, and network congestion, the authors present an initial framework for low-latency end-to-end services. This architecture has been influential in the refinement of the field and suggests offloading of computationally-intensive tasks from mobile devices to the cloud or nearby computational nodes, guiding the development of the modern network edge infrastructure.

Subsequent works on multi-access edge computing focus primarily on delivering a multitude of in-network services to several users simultaneously [66].

2.3.2.2 Fog Computing

The adoption of the cloud was supported by advancements in networking. Despite this, the inherent issues related to unpredictable latency, lack of mobility, and location-awareness remained unsolved. In an attempt to address these challenges, fog computing was devised to provide flexible services at the network edge [154]. The distributed network intelligence proposed by fog computing is a core component of next-generation IoT networks. Below are some of the advantages presented by the architecture:

Data distribution: Collection of data at a centralised location presents scalability issues when billions of IoT sensors are employed. By distributing information throughout the network in-

frastructure, faster data processing can be achieved [90].

Bandwidth conservation: Transferring data across the core of the network to the cloud leads to increased demand in infrastructure capacity. By placing data processing functionality closer to IoT devices, the load on the network infrastructure is reduced [134].

Real-time operation: For a number of applications, IoT device information has to be integrated and acted upon rapidly. Some sensors (e.g., environment sensors) provide feedback that requires action in short timescales. Unpredictable latencies due to transferring information to and from the cloud is a limiting factor in new applications, which can be mitigated through processing capabilities at the edge of the network [142].

2.3.2.3 Edge Network Services

In-network services are a cornerstone for modern networks (e.g., 5G). This functionality is enabled by deployment and operation of VNFs at the network edge. The NetFATE (Network Functions At The Edge) architecture is a representative example [86]. The work allows simplification of function deployment and reduction of operational costs. The solution is limited to devices with specialised virtualisation support, and does not present orchestration algorithms.

Similarly, the GNF (Glasgow Network Functions) framework [34] is another example of achieving in-network function deployment. Use of lightweight containers (e.g., Linux containers, Docker) allows for deployment on a wide variety of devices. However the orchestration mechanism used relies on a computationally intensive optimisation model.

2.3.2.4 Examples of Edge Network Devices

Equipment vendors have begun to offer industrial solutions tailored for the network edge. For example, OnLogic [11] specialises in Edge Servers based on Intel processor technology that allow accessible hosting of VNFs within the network infrastructure. Another provider, Veea[12] builds upon the ARM architecture to offer a wide range of network boxes that can be deployed throughout the network, that can run services close to end users.

The availability of commercial hardware that can enable heterogeneous NFV support at the net-

[11] https://www.onlogic.com Retrieved May 2020
[12] https://www.veea.com Retrieved May 2020

Device	Release	Architecture	CPU	RAM
Residential CPE Routers				
BT Smart Hub 2	2018	ARM Cortex-A9	2x1GHz	N/A
Virgin Hub 4	2020	Intel Puma	2GHz	1GB DDR3L
Google Fiber Network Box GFRG 210	2016	ARM v5	1.6GHz	512MB
Commodity Routers				
NetGear D7000	2016	ARM	2x1GHz	256MB
Ubiquiti Dream Machine	2019	ARM Cortex-A57	1.7GHz	2GB
IoT Gateways				
Dell Edge Gateway 3003	2017	Intel Atom	1.46GHz	2GB DDR3
HPE EdgeLine EL4000	2016	Intel Xeon	4x3GHz	64GB

Table 2.1: A selection of Edge Network Devices according to [36]

work edge indicates a shift from cloud and data centre-based services towards services located in close proximity to end-users. Further evidence for this is the evolution of customer edge devices, with their functionality expanded to provide added value for customers. Advanced services, such as parental control filters, bandwidth management, network storage, and network-wide Virtual Private Networks are being provided.

Customer Premises Equipment (CPE) provided by Internet Service Providers, as well as commodity Routers, give access to advanced hardware functionality that can be applied to in-network service provisioning. We enumerate a selection of these devices in Table 2.3.2.4.

The majority of residential CPE and commodity home routers include multi-core ARM or Intel processors, up to 2GB of RAM, have support for Linux-based Operating Systems (e.g., OpenWRT [43], DD-WRT[13]) and have been shown able to operate simple network functions [36].

Furthermore, commercial IoT Gateways, which connect multiple physical sensors to the wider Internet, are becoming widespread. With a wide range of Intel-based processors and ample RAM, devices such as Dell Edge Gateway 3003 or Hewlett-Packard Enterprises EdgeLine EL4000 can be used for in-network analytics of network data and provide security services for the IoT devices connected.

We expect all of the devices presented above can be used alongside network-wide VNF servers as part of the infrastructure required for distributed network service provisioning.

[13]https://dd-wrt.com Retrieved November 2020

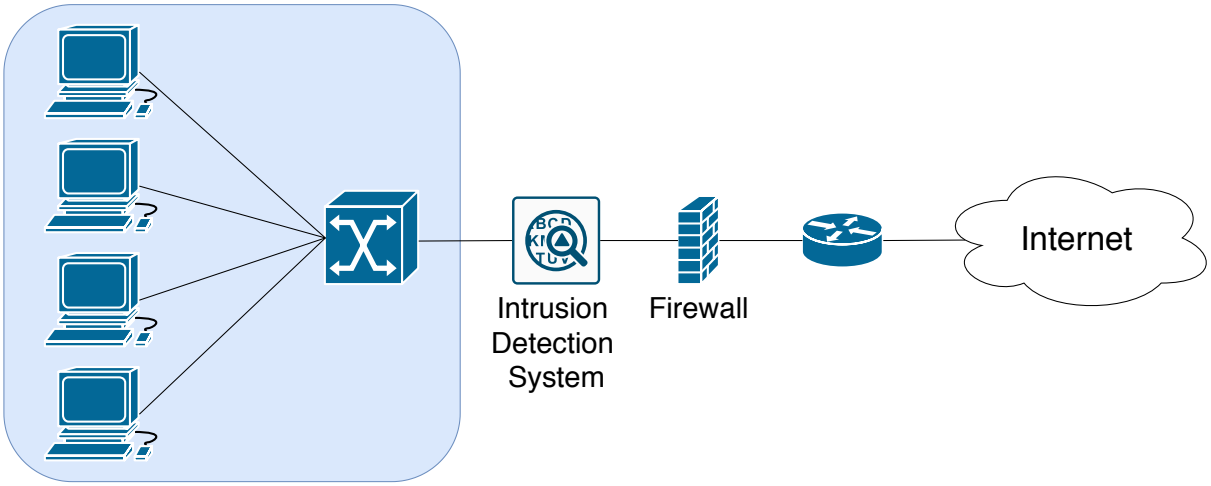

Figure 2.3: A typical security deployment (e.g., for an enterprise network)

2.4 Network Cybersecurity Principles and Applications

As presented in Sections 2.2 and § 2.3, modern networks have evolved to play an important role in modern infrastructure. Being of critical importance, they are increasingly prone to misuse and exploitation. Common purposes of network infrastructure attacks are related to accessing, gathering, manipulating user data, or affecting the availability of the overall system [79] [138].

For example, a Denial of Service attack aims to disrupt normal operation of services by over-consuming of computational or network resources [97]. In today's networks, it is one of the most encountered types of attacks, with a substantial increase in frequency and volume [79]. In order to mitigate such threats, operators have been employing network security measures.

In this section, we look at the evolution of security services with respect to the virtualisation of networks and provide an overview of the current challenges encountered in securing modern networks.

2.4.1 Traditional In-Network Hardware Boxes

From the beginning of the widespread adoption of the Internet, providers have been using a combination of prevention, detection and mitigation techniques to enhance resilience of their operational infrastructure [30]. A typical deployment of these services [97] is presented in Figure 2.3.

Prevention has been achieved by limiting access through the network. For example, it can be done through restricting undesired or unsupported applications. Detection of attacks is done through dedicated security components throughout the network that can inspect network traffic for known malicious behaviour. System administrators frequently employ Intrusion Detection or Prevention Systems (IDS/IPS) or anti-malware services that usually perform Deep Packet Inspection (DPI) within the network to achieve this [12]. Finally, mitigation techniques are done based on filtering malicious network traffic [56], but do not guarantee the elimination of an attack. A common technique is upstream router mitigation of detected attack traffic [67]. The high-level network functions required to implement the network behaviour are presented below:

- **Packet Header Inspectors:** A packet header inspector function (e.g., firewall, access control list, or network address translator) is used to allow or deny traffic based on simple network header information, such as IP address, protocol, or port. It is commonly applied at the network boundary to examine all egress and ingress traffic through the network [159]. Frequently it is employed for both prevention and mitigation of network attacks.

- **Deep Packet Inspection (DPI):** A DPI service performs more in-depth analysis on network traffic, often inspecting application data to determine the existence of an attack. The two most commonly encountered subtypes are anomaly-based and signature-based DPI. The former models normal behaviour of the network and reports intrusions based on deviation from the model [50]. The latter uses a list of known signatures against which traffic is compared [121]. The majority of DPI services do not run at line rate, and detect the presence of an attack after it has occurred [38].

These network functions saw initial implementation through specialised Hardware Appliances (HAs) that had to be physically located within the network. These HAs had a limit on the number of rules that could used and on the volume of traffic traversing them. The configuration is done through specialised control protocols that are vendor-specific. For example, the Cisco PIX 506E Security Appliance[14] supported up to 25000 concurrent connections.

Providing ICT infrastructure security in the face of increased network demand and adoption of new applications has proved increasingly challenging [6]. Replacement or scaling of services required procurement and installation of new HAs, a process that often took weeks and required significant Capital Investment. Management and re-configuration required time-consuming manual operation.

[14]https://support.hpe.com/hpesc/public/docDisplay?docId=emr_na-c03659458 Retrieved May 2020

2.4.2 Virtualisation of Network Cybersecurity Services

To mitigate the problems of legacy Hardware Appliances (e.g., up-front cost, reduced deployment flexibility, scalability limitations, etc.) software-based security services have been proposed in the late 1990s [121] [123]. These services were designed to be operated on commodity server hardware and software (e.g., as Linux kernel modules or userspace applications).

One notable implementation is the Linux IPTables module [123]. It is a userspace application that allows configuration of IP packet filter rules of the Linux kernel firewall. Ushering in softwarisation of security functions reduced the costs incurred in deployment of additional modules.

A notable DPI implementation for performing Intrusion Detection is Snort [121]. Designed as a software-based packet inspector, it allows for high reconfigurability through software rules, and provides a mechanism for dynamically adding in new functionality in the form of additional software modules.

Although the software implementations for these security functions enabled decoupling of network functionality from underlying hardware, the problems related to service location, scalability and performance remained. Many of the newly proposed solutions were designed as monolithic applications [5], able to use a single application thread for packet processing. In modern networks, where 40 Gbps bandwidth availability is becoming commonplace, these network services are an obvious bottleneck in performance [38]. Their functionality was still bound to the underlying hardware location, and introduction of new network services required installation of physical systems within the network topology [127].

The introduction of SDN and NFV concepts helped overcome the remaining operational challenges by providing virtualisation and expanding the number of devices that are able to host networking services [8]. Through NFV, functions and services can be provisioned and deployed faster, and the problem of vendor-compatibility is reduced [11].

The main challenges in adopting VNFs are related to performance when compared to specialised HAs (where software solutions are generally inferior to their hardware counterparts in terms of processing latency and overhead). To address performance limitations, research projects have proposed new approaches to providing data plane processing. Examples include the Intel Data Plane Development Kit[15] and using network hardware for packet manipulation [159].

An emerging paradigm in design and use of security-oriented VNFs is the departure from the

[15] https://www.dpdk.org Retrieved May 2020

monolithic architecture. Eliminating unneeded packet processing overhead can lead to improvements in throughput and delay. vNIDS [85] provides such an implementation for DPI-based systems, and presents a way for partitioning application functionality into targeted microservices. A similar approach has been proposed by Tourani et al. for privacy-enhancement services [138].

2.4.3 From static to mobile devices

As the shift from HAs to softwarised security services occurred, the network usage began to shift as well. Advances in telecommunications, processor performance, and hardware manufacturing processes allowed for the widespread adoption of mobile and portable devices [128]. Portable devices allow for greater user mobility. This departure from static clients to ones that can roam between multiple networks over time poses challenges with respect to network infrastructure resilience. A user's expectations of an uninterrupted experience when migrating from one network to another require transfer of network services from one location to another [125].

Challenges related to user mobility have been at the forefront of edge-based network services [64]. While the main body of work focused on ensuring service continuity for generic VNFs [154] [134], the techniques developed can be used for provisioning of infrastructure security functions. Cziva et. al [32] propose a migration strategy for providing low-latency network services targeted at roaming users. The work focuses on assigning a threshold for latency violations of specific services, and performing migration to hosts closer to the user once said threshold has been exceeded. To allocate services to the network infrastructure, the authors propose the use of an optimisation problem combined with early termination of the associated implementation based on Optimal Stopping Theory. The approach doesn't take into consideration the migration of inline network services as is often encountered in network security.

The need for mobile-centric systems is further reinforced by the rapid growth in network-capable devices with improper security and privacy considerations [158]. The policy abstractions of existing security services are not expressive and customisable enough to provide adequate flexibility in the face of the emerging threats, and the need for so-called μ*mboxes* is identified to provide the necessary deployment in resource-constrained scenarios.

2.4.4 Context-based security policies

Roaming devices pose further challenges with respect to infrastructure security. These devices can join an untrustworthy network, become compromised, then propagate the issue into other networks [148]. Traditionally, this has been mitigated through restrictive policies that allow only certain types of traffic. These security strategies conflict with the visions of next-generation networks and adoption of new protocols and applications, limiting innovation [111].

Research projects have focused on partitioning of security functionality based on behavioural information gathered from the network. An added benefit is the creation of customised security services for each of the devices in the network. Modelling of device behaviour and functionality has led to the definition of a new security approach. The authors of [18] propose definition of security policies based on expected behaviour for different network devices. The work describes the modelling process, with an example use-case in healthcare environments. The work is further expanded [102] with the use of contextual graphs. However, the works do not describe any implementation details, or any performance-related evaluation.

Li et al. [84] describe a multi-domain approach to orchestration in context-aware networks. In their work, the authors derive the device context based on network location and application traffic. This context is stored in local and global metadata headers for use in multi-domain networks. Orchestration of resulting VNFs is performed at a top level. Fine-grained, network-specific placement is performed by Docker Swarm, a Docker-native clustering system. Operator costs are prioritised over network performance metrics in this work.

Morrison et al. [101] detail a framework for programmable in-network security enforcement. The Poise system has two novel components: a *client module* and a *policy compiler*. The *policy compiler* uses a domain-specific programming language for expression of network security policies based on device information and the state of the network. It outputs a configuration for the *client module*, and a series of data plane processing programs, written in P4, to be deployed on programmable switches that enforce the network policy. The *client module* component is responsible for collecting and reporting contextual information. Its current implementation is designed as a kernel module, which relays the gathered information. The main assumption of this work is the ability of devices to accurately and truthfully report contextual information (e.g., GPS location, screen status, accelerometer readings) to a network operator. Privacy implications aside, the assumption that kernel behaviour cannot be compromised is incompatible with current edge environments. The work is, at the time of writing, in early stages, with preliminary results only related to the overheads encountered by utilising client modules. The evaluation also factors in policy violation detection, but does not provide insight into the resulting process.

The context-based security concept is further refined by Yu et al. The Precise Security Instrumentation [157] project aims at providing infrastructure security in enterprise environments through a specialised controller that redirects per-device network traffic based on intent. The devices are categorised based on contextual knowledge (e.g., internal server, workstation, laptop) and security policies deployed to analyse deviation from expected behaviour. The observed deviation is used to instantiate new services for the given device, to mitigate infrastructure threats and perform peripheral analysis of the network traffic. The deployment and orchestration of such services is not presented within the publication.

TENNISON [44] presents a network-wide event processing and correlation framework to analyse distributed security application alerts. It performs light-weight monitoring of network flows and integrates the information with DPI alerts to provide operators with a global overview of infrastructure threats. The proposed controller provides a southbound interface that collects information from IDS/IPS network functions, SDN switches packet counters and other in-network flow statistics monitors (e.g., sFlow). The northbound interface allows specialised network-wide security applications to run. These applications define device-specific policies that are to be followed within the network and monitor deviation from expected behaviour. TENNISON also integrates a hybrid SDN/NFV mitigation strategy, by informing the SDN controller of the required high-level changes required to mitigate malicious traffic and requesting new security services for the NFV MANO component to deploy.

2.5 Orchestration of Network Functions

Orchestration of network functions encompasses allocation, instantiation, configuration and lifecycle management [116]. One of the main challenges in adoption of NFV is the fast, reliable, scalable, and dynamic orchestration of the associated network functions [11]. In terms of instantiation and lifecycle management, virtualisation support (e.g., Xen, Docker, KVM, etc) can be used to offload the required tasks. Configuration of VNFs is closely linked to the type of service being requested, and is generally provided by network operators.

Server-side properties (such as CPU, memory, IO) and network resources (bandwidth, link load, connectivity requirements) have to be factored in when performing allocation [71]. The latter are highly dynamic and can change over time, as users join or leaving the network or interact with new types of applications.

In general, VNF orchestration is similar to the well-studied problem of Virtual Machine (VM)

orchestration applied in Data Centres and Clouds [72] [139] [33] [31]. However, while taking into account server-side resource, VM placement does not account for network properties that many VNFs interact with. Therefore, they cannot be directly applied for use in NFV domains.

The next two sections detail the most important aspects of VNF orchestration: selection of the servers and allocation of resources for execution of VNFs, and creation of complex network services using multiple network functions.

2.5.1 Resource Allocation and Placement of Network Functions

Allocation of VNFs is an NP-hard optimisation problem, as it is a generalised version of the NP-complete Virtual Network Embedding problem [47]. Algorithms that attempt to place VNFs are divided into two categories: optimal (or exact) solutions, or heuristic solutions.

Exact solutions propose the identification of an optimal scenario for resource allocation. Because of the NP-hard nature of the problem, small instances of problems are used as inputs for these solutions. They are typically used as a baseline when evaluating heuristic solutions. Heuristic solutions do not target an optimal allocation of VNFs, instead opting to minimise execution time when solving large instances of the resource allocation problem. The following sections contain examples of both types of solutions.

2.5.1.1 Optimal Network Function Placement Solutions

Optimal solutions are generally formulated using Linear Programming (LP), or linear optimisation models. These models are a series of mathematical linear relationships that aim to provide the best outcome by maximising (or minimising) an expression. They are expressed in canonical form as:

$$\begin{aligned} \max_{\mathbf{x}} \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\ \text{and} \quad & \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{2.1}$$

where $\mathbf{x}$ represents the vector of variables (unknown, to be determined), $\mathbf{c}$ and $\mathbf{b}$ are vectors of known coefficients, A is a known matrix of coefficients, and $(\cdot)^{\mathrm{T}}$ is the matrix transpose. The expression to be maximised (or minimised) is called the **objective function**.

In particular, Integer Linear Programming (ILP), a form of LP that restricts variables to integers,

is preferred for formulation of placement problems. An integer decision variable can accurately and succinctly express VNF-host mappings.

While non-integer linear programs can generally be solved efficiently in the worst-case [20], ILP problems with bounded variables are NP-hard [88] [103]. Specialised software, called a solver, such as the proprietary Gurobi [57], the open-source GNU Linear Programming Kit (glpk) [105] can calculate solutions for ILP problems. They rely internally on algorithms such as branch-and-bound or branch-and-price [83].

Bari et al. [11] formulate an ILP VNF orchestration problem that aims to determine the number and placement of VNFs that optimises operational costs and network utilisation, without violating Service-Level Agreements (SLAs). An evaluation based on a CPLEX solver implementation is presented, where results show that VNF-based services provide more than four times reduction in operational costs.

VNF-P [100] proposes an exact solution to the placement problem. An ILP for hybrid NFV-environment where general purpose and specialised hardware coexist is presented. The aim of the placement is the reduction of Operational and Capital Expenses through minimisation of the number devices used for provisioning of network services. Because of the scaling limitations of ILP problems, evaluation has been done on a small-scale network with various traffic loads. The algorithms used by VNF-P finish in 16 seconds or less, and are able to react to changing traffic demands.

An approach revolving around network properties is presented by Gupta et al [55]. The authors present a linear programming problem that minimises bandwidth consumed by routing traffic through selected paths. Placement of VNFs is performed at optimal locations in the a programmable network environment. The term “Network-enabled Cloud” (NeC) is also introduced, which describes a cloud environment with extensions provided by programmable packet and optical network nodes. Results presented show a reduction in network resource consumption.

The T-NOVA NFV platform’s TeNOR project [117] presents two models for mapping VNFs to Points-of-Presence within the network. The objective function for the ILP model is a weighted sum of the cost of assigning a VNF to a specific PoP, the sum of the overall delay, and the overall network resource usage.

The model proposed in [32] focuses on placement of VNFs, considering bandwidth and latency constraints to minimise the end-to-end latency. The ILP presented determines placement of

VNFs in edge-cloud environments for multiple users. The focus is on individual VNFs that act as traffic endpoints (e.g., caches, video transcoders, etc). A relocation and migration strategy based on Optimal Stopping Theory (early stopping of execution while getting near-optimal results) is included. Evaluation on simulated real-world networks achieves low user-to-service latencies with minimal Service-Level Agreement violations.

In [13], an optimisation problem for mobile core networks to deploy VNFs is presented. Placement onto nodes of the physical substrate network and optimal traffic routing between nodes is achieved through the work. Similar to many other works in the field, the objective is to minimise the cost of occupied links and node resources. Simulation results over two nation-wide network topologies outperform traditional Virtual Network Embedding optimisation approaches.

The authors in [12] address placement of security services in virtualised environments. The problem is modelled as a Mixed-Integer Linear Programming model (where some decision variables are non-discrete) on the ISP environment, with the goal of minimising costs of network operators. The work proposes reduction of the Points-of-Presence in the network to host desired services. Implementation details are provided in the form of a prototype, with use of the CPLEX solver to generate a solution. The network models used are relatively small, and the reported runtime is less than one second.

2.5.1.2 Heuristic Network Function Placement Solutions

Heuristic, non-exact solutions have been proposed by many researchers as alternatives that minimise execution times for finding placement solutions.

Acknowledging the complexity in optimal placement solutions, Yoshida et al. [156] provides a two-step heuristic for placement of VNFs that considers computing and communication costs. The first step relies on a resource filtering mechanism to determine which resources are eligible for consideration. The resulting output is then provided to the second step, a genetic algorithm, for generating placement decisions. The evaluation of the proposed approach provides lower computation times when compared to a naïve approach and analyses scalability of the algorithms.

In multi-tenant data centres, another genetic algorithm was proposed by Qu et al. [114], which aims to minimise latency of VNF placements by assigning execution time slots to different services. The algorithm performs virtual link bandwidth allocation, and scheduling of VNFs to meet service demands in a centralised manner.

For enterprise wireless networks, Riggio et al. [118] propose a management and orchestration framework compatible with the ETSI NFV specification. The allocation algorithm uses a heuristic-based approach that optimises VNF placement based on application-level requirements (e.g. latency). Evaluation, performed using a simulated network topology, aims for high service acceptance ratios and minimised server and link utilisation.

Using a real-time service graph mapping, Nemeth et al. [104] propose an allocation algorithm for carrier networks. The proposed environment shows a very large number of VNF placement requests arriving within a few seconds. By using a greedy backtracking method, the authors aim at minimising the time required to derive a placement of VNFs. To enable flexibility, this solution allows for fine-tuning of parameters that influence the quality of orchestration and enable desired acceptance ratios.

2.5.2 Network Function Chaining

Beyond the need to place individual VNFs within the network, operators are looking to add value and enhance their resilience by providing complex services composed of sequences of VNFs (also called Service Function Chaining (SFC)). One traditional example is a chain consisting of a Firewall, an IDS, and a Wide Area Network Optimiser, that aim to filter out unneeded traffic at the network core and increasing network availability. This section details notable solutions to the VNF Placement Problem that allow for placement of function chains at multiple network locations.

Pham et al. [110] proposed SAMA, which proposes service function chaining that minimises operational and traffic costs. The problem is initially formulated as an ILP, and a heuristic using a Markov approximation algorithm based on sampling is provided. The goal of the algorithm is to reduce infrastructure utilisation and network costs. Furthermore, the authors present a subproblem division that can enable distributed operation within the network infrastructure.

Vizarreta et al. [144] propose a cost-focused ILP approach that minimises estimated Capital and Operational Expenses. As a result, the QoS aspect becomes a secondary objective in their proposed model, and is impacted by other factors within the network.

Luizelli et al. [89] put forward a solution for Service Function Chaining between multiple regions by utilising the NFV Points-of-Presence (PoP). The authors describe an ILP for chaining of branching SFC that can accommodate inter-regional traffic, with the goal of minimising the number of VNF instances allocated within the infrastructure. Furthermore, a heuristic algorithm

for guiding the search is proposed by bounding the search parameters, obtaining near-optimal results.

In [71] a formulation for optimal use of network resources is provided for the placement of VNF chains. Their simulation results show how the ILP approach can accommodate more flows and reduce the overall network resource usage. However, the presented simulation only features two types of SFC requests, with no operator-defined composition strategy.

Kuo et al. [80] look at the relation between link and server usage in VNF allocations for enterprise networks. The goal of their solution is to maximise the number of accepted SFC requests, which requires placement of the VNFs within the network and generation of appropriate steering paths for the chains. In achieving this, the authors consider link requirements of the flows and VM capacity within the infrastructure. To obtain the relationship between link and server usage for a request the authors propose an additional ILP that aims to avoid deployment beforehand.

In terms of allocating security-centric VNFs, Ali et al. [5] propose a different method to tackling the allocation problem, being presented as a variable-cost variable-sized bin packing problem (VSBPP). Targeted towards multi-tenant data centre environments, the placement is influenced by server resource availability and aims to increase overall requests. The solution proposed does not take into consideration QoS metrics. Evaluation of the model shows an increase in request acceptance rate (the number of services that can be hosted on the NFV infrastructure) and reduction in unsatisfied server resource requests.

Finally, in [41] the progressive provisioning of security services (PESS) model is presented. It estimates the processing delay based on residual computing resources and factors it into allocation of services. The objective function aims to minimise cumulative usage of physical resources. Acknowledging the limitations of an ILP formulation, the authors also propose a heuristic solution. Both approaches propose sharing of security services between multiple users, which increases related management overheads when user migrations occur within the network. Furthermore, the evaluation is performed with the main goals of minimising residual bandwidth of the networks and VNF host CPU availability. The resource availability of the aforementioned hosts is akin to that of a TSP's core infrastructure, with limited consideration towards edge networks.

2.6 Summary

This chapter studied the evolution of programmable networks, from the early adoption of the Internet in the 1990s to modern times. As described in Section 2.2, SDN and NFV allow operators to manage their networks in a programmable, dynamic, and customisable way. SDN provides a centralised control plane for networks, while NFV enables flexible creation and use of services at multiple network locations.

Section 2.3 presented the current state-of-the-art network paradigms and migration of services from centralised, cloud and data centre-based infrastructures closer to the end users, that free up core network infrastructure, provide location awareness, and provide low-latency network services.

Section 2.4 provided an overview of network security approaches that have been used to increase the resilience of infrastructure and protect end users. Specialised hardware appliances (HAs) have been discussed, presenting challenges regarding lack of deployment flexibility, high operational and capital expenses, limited extension of functionality, and inefficient management of resources. Softwarised security services were introduced to alleviate some of the identified problems, later being applied to the NFV environment, with their potential for use in enterprise and data centre environments investigated. The current paradigm of device-oriented security obtained from contextual information is presented and how adoption into enterprise networks has been achieved.

Orchestration of network services in general, and security functions specifically is a complex process. Section 2.5 reviewed the two principal solutions: exact placement problems and heuristic-based algorithms. It outlined the limitations of using exact solutions on large-scale networks and the trade-offs made for deriving appropriate solutions that match dynamic network conditions. The section concludes with the required considerations for creation of complex services through chaining of multiple network services.

Based on the work presented in previous sections, the following directions for this thesis are identified:

1. The benefits of the network edge allow for use and deployment of lightweight VNFs that can be dynamically started and stopped. However, the current security-centred VNFs are designed as monolithic, heavy applications that require significant computational availability in their operation. Therefore, there is a need for lightweight, microservice-based

security services that can operate on low-cost edge devices.

2. Extending previous orchestration work, a dynamic, low-latency placement strategy for inline VNFs is required to manage the aforementioned security VNFs according to dynamic network properties and user expectations to prevent and mitigate undesired malicious activity within the network infrastructure.

3. Long-term lifecycle management of network services needs to correspond to user behaviour. For example, because the security services filter undesired traffic, de-allocation of network services when no longer required or upon user departure are essential in providing flexible user services and an enhanced experience.

In the next chapter, we further analyse the constraints imposed by the network architecture, service types and user expectations. We synthesise the design requirements for network security microservices and associated orchestration framework.

Chapter 3

Design

3.1 Overview

Edge networks have been enhanced over the years to support the emerging needs of Internet applications and services, ranging from VoD hosting to general computation clusters, each with different security requirements [149] [138] [40]. For instance, a typical IoT Gateway may require modules that detect and mitigate DDoS flooding attacks and remote code execution, while critical servers may require a firewall, IDS/IPS and DPI to guarantee high availability and network integrity. However, these security services are limited by the computational power needed to process network traffic, restricting deployability and posing constraints on flexible management of network functions.

As discussed in Section 2.4, the existing Virtualised Network Functions (VNFs) that provide infrastructure security functionality adopt a monolithic software architecture that requires plentiful computational capacity. Attempts to provide distributed, network-wide functionality are generally limited to specific behaviour. However, these solutions do not cater to the environments of next-generation networks, with functions operating on real-time network traffic and a goal of providing transparent network services through minimisation of packet processing overheads.

To address these limitations, this chapter presents selected design considerations enhancing the generic ETSI NFV reference architecture. The chapter presents the motivation and high-level system requirements for use of security-oriented VNFs at the network edge in Section 3.2. In Section 3.3 the main design considerations posed by the presence of lightweight, composable security functions are presented. Then, in Section 3.4, a comparison between lightweight secu-

rity VNF architectures, traditional software solutions and legacy hardware appliances is made, alongside a justification for the use of dynamic creation and composition of microservices in our framework. Management and orchestration concepts are discussed in Section 3.5, with consideration given to orchestration of services by adhering to Context-based security principles, and operation in resource-constrained environments. Finally, Section 3.6 presents the fundamentals for dynamic, latency-optimal on-path placement orchestration of Network Service Chains that is required to efficiently place and re-allocate services due to movements of users or changes in network behaviour (e.g., congestion, link failure, etc.).

3.2 System Requirements

This section provides motivation for a new security NFV architecture by detailing the need for lightweight customisable security functions, autonomous lifecycle management strategies, and on-path, low-latency placement orchestration. The section then presents the core design requirements and operational principles to be used in production networks.

3.2.1 Security Service Flexibility

Network Function Virtualisation has been introduced in the 2010s by large telecommunication service providers to virtualise in-network services by using commodity, yet powerful, x86 servers and network cards. Instead of using proprietary hardware appliances for providing network security (such as firewalls, intrusion detection systems, deep packet inspectors, etc.), a more flexible service management approach using popular system virtualisation platforms (e.g., Xen, VMWare ESXi, KVM, etc.) was adopted. As outlined in Section 2.5, the initial implementations were focused on provisioning large VMs in Data Centre networks to process large amounts of traffic in the core network and co-host multiple VNFs.

The approach has recently seen further refinement with offloading of VNFs to the network edge, to better cater to the emerging requirements of mobile and IoT devices. Providers have begun enhancing the network edge with smaller Data Centre-like infrastructure (often called cloudlets) and virtualising the existing processing hardware. This shift, while it provides access to devices in close physical proximity to end-users, brings with it the less powerful computational environment for use, from commodity routers, to IoT gateways, to enterprise edge servers.

The overhead incurred by controlling the devices needs to be kept at a minimum, especially

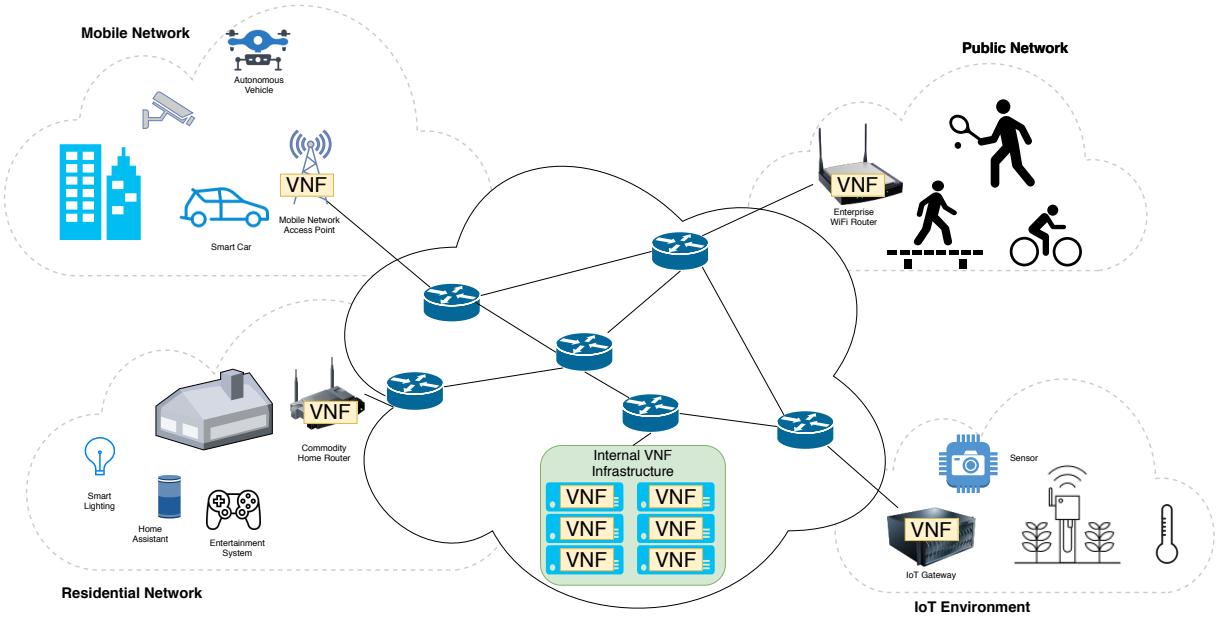

Figure 3.1: Edge Network Environment

given the dynamic contexts in which mobile devices are present. Due to resource availability concerns, and considering user mobility, having increased autonomy in the lifecycle management of the VNFs is paramount, without sacrifices with respect to flexibility in general orchestration. The context-based security paradigm described in Section 2.4 offers the logical decoupling of security policy (the expected parameters for an application or device to function) and implementation, however the details of management and orchestration strategies conforming to the lightweight nature of edge networks have yet to be thoroughly defined.

As presented in Section 2.4, traditional virtualised security modules have not yet adapted to the lightweight requirements of these emerging networks. Monolithic software architectures (e.g. Snort IDS [121], Zeek [137], Netfilter Firewall [123], etc.) are still considered the most popular solutions for providing security functionality. The processing overheads incurred as every packet traversing monolithic services quickly outweighs the benefits of proximity to end users. Furthermore, the services often perform duplicate functionality (e.g., IP protocol validation, L4 decapsulation and re-encapsulation) that increases the delay.

To solve the challenges presented, the designed system will explore the options for lightweight, microservice-based VNFs to enable creation of minimal-overhead services that can be run on lower-power devices. The management and orchestration components will investigate the use of autonomous lifecycle management while providing lightweight NFV functionality for use in resource-constrained environments, as shown in Figure 3.1. Finally, placement of services within the network with a minimal end-to-end latency objective will be explored.

3.2.2 Placement of Security Service Chains

The allocation of VNFs to physical servers (also known as the VNF Placement Problem or VNF Embedding) has been one of the most studied topics within the last few years. Solutions coming from academia have been presented for optimal and heuristic placement, while industry solutions make use of simple, predictable heuristic schedulers (e.g., Kubernetes' resource scheduler, OpenStack's weight-and-filter scheduler). Many of these placement solutions stem from the VM Placement Problem that arose when virtualisation and Cloud environments became widespread. The placement of VNFs in edge environments presents several challenges.

Foremost, the current approaches to VNF placement focus primarily on minimisation of physical servers used for hosting, as presented in Section 2.5.1. This approach can be translated into reduction of capital and operational expenses, as fewer servers have to be purchased, managed, and reduces energy consumption. This objective is relevant within the core network, but for edge networks where existing infrastructure can be leveraged, the concerns mentioned become less applicable. The design of a placement strategy at the network edge needs to focus on low-latency services that allocates VNFs on the path between an end-user and the intended recipient of network traffic.

Moreover, many of the placement algorithms do not take into account the generation of complex network services through chaining of multiple VNFs. The recommendations made by the IETF for deployment of network functions in Mobile Networks showcase the need for chaining multiple VNFs in order to meet the security and service requirements of both telecommunication service providers and end-users. The design of a VNF resource allocation algorithm needs to take into account multiple network functions chained together. The allocation can be co-located on the same physical server or span multiple locations within the network, while ensuring that the routing of network traffic reaches the intended destination.

For evolving networks, Context-based placement, which assigns a VNF to a single user, has shown promise in previous research. Adoption of the paradigm at the network edge leading to utilisation of residual network resources is to be explored as it can provide reduced end-to-end latency and reduction of Capital and Operational expenses by avoiding provisioning of large-scale Data Centres within edge netwokrs.

Finally, the orchestration algorithms do not take into account variations in network dynamics or user relocation. These variations can arise from individual link congestion, latency fluctuations, or link faults. The changing network conditions influence the user, and the design of the placement mechanism needs to take them into account.

The design looks to provide a latency-optimal placement and routing solution for on-path network service chains. Such orchestration algorithms can rely on real-time network telemetry, such as topology information and latency measurements, along with QoS requirements defined by the operator to identify the optimal placement of network function based on temporal dynamics of the network. In the era of network slicing and variable performance requirements, the guarantees offered by latency-optimal placement of VNFs at the network edge become a cornerstone requirement for emerging applications.

3.2.3 Operational Principles

The platform allows operators to define new security policy elements, and create, deploy, and manage on-path services on-demand. The increasing complexity of modern networks makes static deployments of network functions, through traditional hardware appliances, difficult to scale and increases capital and operational expenses. Autonomous functionality of the NFV orchestration platform should be easily accessible for configuration and customisation by network operators.

Operation is designed to support a heterogeneous infrastructure, with a wide spectrum of devices. As presented in Figure 3.1, the targeted infrastructure consists of IoT gateways that connect sensors to the Internet, low-cost home routers that can be found in residential environments, enterprise edge servers overseeing an entire organisation's network, radio cells servicing multiple mobile devices, and autonomous vehicle beacons that facilitate communication between multiple traffic participants. The proposed system should offer deployment on each of these device classes, to bring programmable network services as close as possible to end users.

Detection of new devices or applications needs to be performed autonomously by the network, with minimal operator interaction. In environments where users roam between multiple network access points, operator intervention on deployment of new network functions needs to be performed automatically. To this end, the designed system should integrate with existing network management elements (e.g., SDN controllers) to detect the presence of new devices on the network and perform the necessary deployment steps without manual intervention from telecommunication service providers.

Customisation of lightweight VNFs is done through code. The modification of behaviour in one VNF is done programatically, with the downside of incurred communication costs, instead of manually editing properties through graphical user interfaces. This approach allows for rapid addition and integration of new functionality, while maximising the potential for creating tailor-

made network functions with minimal packet processing overhead. Network operators can automate the creation of new VNFs within the system.

Presence of security services within the network should not impact user experience. Without sacrificing operator principles and best practices in terms of cybersecurity, end-users should experience minimal disruption to their normal activity. The overall design of the system should provide the transparency (from the end-user perspective) by respecting the end-to-end argument [124].

3.2.4 High-Level Requirements

To reach an architecture that conforms to the needs of modern edge networks, the related literature has been reviewed, and current NFV platforms have been studied. Based on the findings, the following high-level design requirements have been identified:

1. VNFs should run on multiple underlying devices, with a primary focus on low-cost hardware that can be found at the network edge or residential premises.
2. Conform to context-based security principles, by assigning VNFs to individual user traffic, promoting modularity and customisability, and allowing reconfiguration in short timescales (a matter of seconds [36]).
3. Placement of VNFs should be done in real-time in order to meet the expectations of users joining the network.
4. Allow individual servers to autonomously perform lifecycle management of VNFs, to minimise control plane overheads and maximise hosting resource availability, but perform reporting of utilisation and health for accurate centralised orchestration.
5. The platform should perform flexible, latency-optimal placement of service chains (containing multiple VNFs) based on temporal network properties and location of end users, a crucial component when services are used at the edge of the network.
6. Support for roaming users should be achieved by migrating or recreating VNFs when the access points change, to provide an overall improved end-to-end (E2E) latency.
7. Transparent traffic routing should be used to provide resilient network connections and allow for replacing individual VNFs without affecting users' traffic.

3.3 Design Considerations

This section introduces the principal design considerations for security-oriented lightweight VNFs and the management and orchestration framework component to enable functionality in edge networks. These considerations are based on, and extend, the general considerations introduced in the ETSI NFV standard [42] and the IETF draft analysis on use-cases for Network Function Virtualisation in mobile networks [59] [58].

3.3.1 Elasticity, Scalability and Responsiveness

Elasticity should be treated as a first class concern in a dynamic network environment [154]. In essence, elasticity is the ability of a management and orchestration component to adapt to workload changes by automatically provisioning and de-provisioning of resources [138]. As an example, users employing new applications require creation of new service chains with minimal operator intervention. Furthermore, when the individual load is high, a VNF should be able to migrate from a low-cost edge device to a more powerful server. This envisioned elasticity can be achieved by using microservice-based VNFs that allow rapid instantiation and deconstruction, low computational complexity, and providing mechanisms for temporal resource utilisation monitoring to perform task migration.

A scalable system can reduce capital and operational costs that a network operator incurs from using an NFV platform within their infrastructure. Scalability of a NFV system is related to the number of servers it can manage and use for VNFs, and the number of users that can be serviced by VNFs [44]. In softwarised networks, the principal component that influences scalability is the amount of traffic present in the control plane. When designing lightweight NFV platforms, servers that autonomously manage some of the VNF lifecycle aspects can significantly reduce the overhead of control plane communication and state required by the NFV orchestrator.

Responsiveness of NFV orchestration should be a high priority when considering dynamic, edge networks [95]. The time from a user joining the network to the availability of given network functionality should be kept to a minimum [157]. Rapid start and stop times for VNFs, rapid allocation of VNFs to physical servers, timely creation of network traffic steering routes, and a mechanism for creating and propagating customised VNFs to servers are the quintessential design concepts for a responsive NFV platform.

3.3.2 Performance in Edge Networks

Performance varies by the VNF type considered. As an example, an on-path real-time Deep Packet Inspector (DPI) VNF has strict requirements to minimise the introduced packet processing overhead, while e.g., a VNF deployed to capture packet traces for offline analysis needs fast data storage capabilities.

Runtime performance of a VNF depends on the software artifacts used, as well as the amount of resources allocated to the VNF. As the devices used in network edge scenarios generally have limited computational, memory, and storage resources, the focus for improving the runtime performance should be on the software artifact, by e.g., creating bespoke VNFs targeting desired behaviour. Performance analysis related to this topic should focus on metrics related to packet processing overhead, VNF reconfiguration duration (time taken for a change in behaviour to occur), time for activation (time required for the VNF to process network traffic).

In terms of management and orchestration of services within the network, the activation of service chains, consisting of multiple VNFs, is an important factor when measuring the performance of a NFV framework. As an example, the time required to determine, allocate, and route traffic through VNFs when a new service request is encountered is a core performance metric in evaluation. The concept, also called VNF provisioning time, is usually measured in minutes in today's NFV frameworks, while lightweight platforms are reducing this time to a few seconds [36].

VNFs and the underlying server management software often need to communicate with the network-wide manager component in order to issue updates, provide utilisation metrics, or perform starting and stopping of functions. Therefore, it is important to provide efficient, rapid communication between VNFs and NFV managers.

3.3.3 Management of large Service Chains and Tiers

In the ETSI NFV specification, management and orchestration revolves primarily around determining the placement of VNFs on to physical servers and managing VNF lifecycle. As discussed, additional concerns regarding performance management, service availability, and infrastructure fault resilience need to be accounted for in the management software. When considering thousands of small, lightweight VNFs that create complex network service chains and tiers in a distributed, heterogeneous infrastructure with hundreds of edge devices and users, additional

challenges need to be overcome.

One of the major contributions of NFV is the ability to rapidly respond to changes in the underlying network infrastructure. As a result, the management component should perform dynamic re-orchestration of deployed services upon detection of a change network properties. As presented further into the thesis, this task runs the risk of becoming computationally complex when many users are involved, no longer able to conform to the real-time orchestration requirement defined in Section 3.2.4.

For security-centric use cases, monitoring of alerts and warnings issued by individual functions is an integral part of the overall functionality. Alongside performance monitoring to report utilisation information, the management component of the NFV platform requires a security event monitor. To provide the functionality, the server-based NFV component needs to relay the information to the management software using carrier-grade techniques, the latter requiring support for handling a large number of simultaneous connections.

Adhering to the Context-Based Security paradigm presented in Section 2.4 depends on the availability of replacement VNFs that can be deployed when a user's network context changes (e.g., when an application behaves unexpectedly, or a device performs operations outside its intended utilisation). To this end, a method for interpreting various alerts issued by the VNFs is needed, and an associated strategy for handling and mitigating various emerging infrastructure security threats. To integrate such behaviour, the operator requires a flexible and extensible control interface to define security policies, their mitigation strategies and associated VNF behaviour that can be composed. Thus, the management software requires a well documented and platform-independent API with platform and language-independent data structures that can describe the desired deployment.

3.4 Microservice Network Security Architecture

As described in Section 2.4 and Section 3.2, security VNFs have been traditionally implemented based on monolithic software architectures. However, the shift towards the use of the network edge for enhanced service delivery is coupled with the limited computational resource availability of servers. Emerging work has been focused around providing application partitioning solutions for running specific network functions in distributed and edge environments, similar to the development of bespoke systems. At the same time, the microservice architecture has been promoted because of the rapid deployment capabilities, reduced resource consumption,

higher flexibility, and platform flexibility. In this section, the major differences between these approaches are compared, and the use of microservices for our network security architecture is justified.

3.4.1 Comparison with Monolithic and Bespoke Systems

Monolithic Systems are designed as a 'one-size-fits-all' solution that can be used in many different scenarios. They encapsulate functionality for multiple types of behaviours, that can be activated at runtime through modification of configuration files [121] [123]. The generality of the architecture used means that resulting software artifacts are highly dependent on their configuration in terms of resource usage, providing uncertainty in VNF provisioning. Furthermore, the configuration files require significant amounts of time to be read and transformed into internal data structures, making VNF provisioning times unnecessarily long. Finally, the packet processing overhead introduced by the dynamic, runtime reconfigurable element, limits the availability of such VNFs for applications that are latency sensitive.

Monolithic architectures require hosting at a specific location. Built on the assumption of global network visibility, the distribution of multiple instances within the network is difficult to achieve, especially when such systems correlate behavioural information from multiple users. Example applications include Snort [121], the de-facto network security service, and its' modules (e.g., Intel DPDK integration, anomaly detection, malware scanner), Zeek [137] (formerly Bro IDS), or Suricata [106].

Bespoke Systems , or specialised distributed VNFs, are specially crafted versions of network functions that distribute the behaviour throughout specific network environments. Bespoke systems are usually derived from existing implementations of VNF classes (e.g., firewall, Intrusion Detection) [1] [2]. A notable example is the vNIDS research project [85], which uses application partitioning to enable Deep Packet Inspection (DPI) on multiple VNF hosts located within the network. While this approach provides lower overall packet processing overhead and allows functionality to be present within multiple network locations, it still requires a logical centralisation of information through intra-VNF communication, restricts integration with Context-based Security systems, and suffers from the same provisioning delays.

Microservices Figure 3.2 compares the architecture and behaviour of Microservices with that of Monolithic Systems. The main advantage in Monolithic Systems is the ability to dynamically enable desired behaviour at runtime, while Bespoke Systems and Microservices require built-in behaviour that is compiled into the software artifact, with static functionality within the VNFs

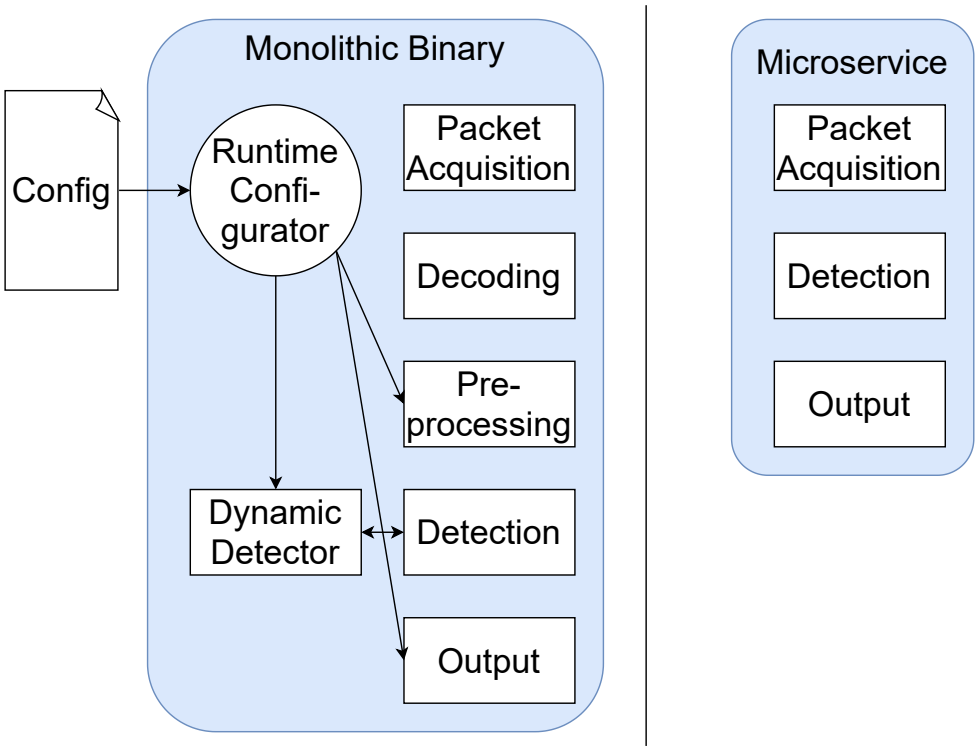

Figure 3.2: Comparison between elements of Monolithic and Microservice VNF Architectures

lifetime. To this end, monolithic applications employ additional elements (e.g., decoding and pre-processing) to achieve flexibility, in detriment to processing overheads. In the case of Bespoke systems, VNFs can be enhanced with new types of the same behaviour (e.g., a new packet signature), while Microservices allow for rapid deployment and execution at the cost of regeneration of software artifacts (e.g., from source code) for any behavioural changes made [92].

Microservices are a good compromise between the general flexibility given by monolithic architectures and partitioning properties of bespoke systems. They allow creation of different VNFs with targeted behaviour from a repository of components, while being able to run independently [69]. At the same time, they incur significantly lower packet processing overhead because of the reduced dynamic behaviour [132]. Because of the predictable system resource utilisation microservice architectures allow for more precise placement on physical servers and higher density of network functions on hosts, at the cost of runtime configuration capabilities [36].

3.4.2 Drawbacks for Microservice Architectures

While microservices provide several benefits when used on resource-constrained edge devices, there are certain challenges that need to be considered when adopting this design for VNF im-

plementations [69].

When envisioning a large number of VNFs operating in a large network infrastructure, the complexity of managing the deployment increases. As every VNF is an independent service, communication between elements has to be carefully handled. In some scenarios, management of VNFs can cause complications because of the incurred latency overhead.

Several research works emerged in order to provide marshalling of smaller VNFs. In order to better utilise computational availability and thus maximise potential throughput of a function, consolidation of services has been previously proposed [76]. This approach risks inadvertently creating monolithic functions that can add unexpected latency overheads when under demand.

Several authors have identified the inherent complexities associated with composition of new VNFs from a series of independent modules, including function description [73], chaining [138], and customisation [85]. Accurate description when performing composition is one such example. According to John et al. [73], these description methods need to cover both the service-level and hardware requirements. This topic has been under continuous investigation within the VM and VNF placement problem space, with models involving simple approximations of hardware availability (e.g., as encountered in [32], [39], [30], etc.), with recent works starting to consider modelling requirements across multiple dimensions, as evidenced by et al. [108]. However, this thesis does not attempt to provide a unified, formalised method for expressing these requirements.

Tourani et al. [138] identifies microservice chaining as a challenge in the security domain, due to the increased risk of vulnerability within the network. To overcome this, the authors envision a secure API that enables isolated inter-process communication between VNFs. Several industry-standard solutions are already available for this purpose, with a notable solution being based around the gRPC protocol[1] which allows for well-defined communication interfaces that are checked during compilation, and enable secure communication using Transport-Layer Security (TLS). Other solutions rely upon REpresentational State Transfer (REST), Socket, Simple Object Access Protocol (SOAP), which are more loosely defined and require explicit knowledge and manual verification of implementation.

Another open challenge, highlighted by Li et al. [85], is the difficulty of customisation. In particular, the authors focus on Network Intrusion Detection Systems (NIDSes), however the challenge remains present within the broader scope of security-centric VNFs. Modification of a single component in a monolithic system requires rebuilding and redeployment of the entire

[1] http://grpc.io/ Retrieved November 2021

system, posing a challenge in highly dynamic edge environments.

In our model each resulting VNF needs to be compiled, requiring a sufficiently powerful server that can perform the task. In the case of multiple simultaneous creation requests, the challenge of adequate VNF provisioning time becomes apparent.

3.5 Edge-based Security Service Orchestration

As described in Section 2.3, the network edge provides unique opportunities in placement of services close to end-users, reducing delays and optimising infrastructure utilisation. However, the network edge consists of resource-constrained devices, as illustrated in Table 2.3.2.4; management and orchestration of network services within this environment must take into account the limited computational resources, and utilise the underlying infrastructure efficiently. In this section, we outline the considerations in designing an orchestration framework for security services at the network edge, focusing on improving resource utilisation.

3.5.1 Usage Driven Services

As evidenced in Section 2.4, the “one-size-fits-all" paradigm for network security is quickly becoming outdated. With the rapid increase of devices, volume of network traffic, and infrastructure sizes, existing management strategies are difficult to maintain and lead to periods when operator best practices are not maintained (e.g., during service updates). Furthermore, mitigation times for emerging security threats are increased due to the delays in reconfiguring the services involved.

Context-based security deployments [157] overcome this limitation by proposing per-device security services that can be logically managed without interference for other users. The above-mentioned limitations can be addressed through this logical separation. Instantiating per-application network security VNFs can enable operators to provide security services only for applications that are being used, thus minimising resource requirements within the network system.

Combining per-device with per-application security paradigms can ensure that operator best practices are followed, while allowing users to maintain network availability when undesired usage is present. Deployment at the network edge also leads to a decrease in core infrastructure bandwidth utilisation for benign purposes, allowing for faster network expansion.

3.5.2 Device Detection

The network edge is inherently unpredictable, with a multitude of devices joining and leaving the network within short timescales. Even in a simple residential environment, there are no unified mechanisms for when devices join or leave a network. As an example, assigning IP addresses within a network requires application layer support through the Dynamic Host Configuration Protocol; the client requesting an address must broadcast a special DHCPDISCOVER message, instead of being discovered by the network. Because of these disparate mechanisms and the fact that initial design of network mechanisms did not foresee mobility, the deployment of services through NFV mechanisms when a device joins the network, or when a new application flow is initiated, becomes a complex management and orchestration challenge.

The flexibility introduced by Software Defined Networking (SDN), which allows a global overview of the underlying infrastructure and connected devices, can be leveraged to overcome this challenge. Using SDN, new devices can be rapidly identified within the network, allowing operators to rapidly employ appropriate security modules which respect their policies and best practices. Being built on top of SDN, the NFV framework can provide integration for deployment and management of these modules, able to react rapidly to changes in device behaviour.

3.5.3 Resource-constrained Infrastructure

The vast majority of devices within the network edge, as presented in Table 2.3.2.4, have limited computational resources that can be used for hosting services. In order to minimise capital expenditures, the capabilities of these devices must be leveraged efficiently in providing additional network functionality through NFV.

The services are managed according to device and application traffic. However, instantiating every potential service within the network without being actively employed leads to inefficient utilisation of resources.

The other aspect presented in Section 3.5.2, in which there is no mechanism through which the network is notified by a device disconnecting, leads to services being instantiated but not used. To alleviate this issue, we propose an autonomous VNF lifecycle management scheme where the VNF hosts decide, based on the existing network traffic, on lifecycle management of individual network services. This allows for the reclaiming of resources, improving the overall ability for service utilisation.

3.6 Latency-Optimal In-line Service Chain Placement

Following on the rationale presented in Section 3.2.2, this section introduces the latency-optimal in-line service chain placement model designed for the orchestration of complex network security services in the proposed framework.

As evidenced in Section 2.5.1, the problem of resource allocation and placement of Virtual Network Functions within networks has been widely studied. The formulations and solutions presented do not present the required features for deployment of *security-focused* VNFs at the network edge. We further present these features and analyse how existing placement formulations accommodate them.

- **Source-to-destination** placement: The formulation must provide placement of VNFs without disrupting traffic flow between communicating devices. Thus, services are placed on the network path between the source and destination. The Nestor ILP formulation [39] and it's multiple resource dimension formulation [108] enable this by identifying and placing VNFs at network vantage points (e.g., VNF Points-of-Presence, Data Centres) and perform traffic routing to ensure that flows transit these intermediate hops before reaching their destination; similar functionality is presented by Doriguzzi-Corin et al [41]. On the other hand, the formulation presented by Cziva et al. [32] does not consider placement of VNFs between source and destination, instead treating the VNFs as traffic end-points (as is common in e.g., CDN services, or Video Hosts) that directly communicate with the source device.

- **Service chaining**: Providing security for a network flow is achieved by employing several specialised VNFs (e.g., Access Control, Firewall, Intrusion Detection) to inspect and process traffic. As a consequence, placement solutions have to support service chains composed of multiple VNFs which can be placed at different PoPs by providing connectivity and traffic steering between VNFs. Monolithic, one-size-fits-all, VNFs perform all of the functionality in a single location, which increases server resource demand (e.g., CPU, memory and throughput) and succumb to scalability limits with respect to traffic processed. The solution proposed by Cziva et al. [32] only performs individual VNF placements, service chains are not supported, which leads to security services being exclusively defined as monolithic applications. Nestor [39] and PESS [41] provide placement for network service chains.

- **Granular security functionality**: The Context-Based security paradigm that is advocated in this thesis is built upon the understanding that multiple flows, even if part of the same

application, have different security requirements and, thus, utilise different service chains and security VNFs. For example, a CCTV system has three network flows: a video stream captured by the cameras, and bi-directional control and management traffic (e.g., remotely control camera pan, tilt, zoom, etc.) For the former flow, a security chain would involve a firewall to fend against information leakage, and the latter flows are serviced by chains that contain firewalls and IDS/IPS VNFs with different goals: the CCTV system needs to be protected against IoT Command-and-Control attacks such as Mirai, and the remote accessor must be defended against web exploits. The PESS solution [41] provides partial implementation by recognising the need for three service chains, but resulting deployments utilise monolithic VNFs which service all of these flows simultaneously. Nestor [39] allows definition of multiple service chains which can satisfy the granularity of VNFs, and the formulation by Cziva et al. [32], because it does not incorporate end-to-end traffic, does not support this scenario.

- **QoS-driven**: The biggest advantage gained by placing VNFs at network edge is the improvement in the Quality of Service achieved by reduced latency. Treating a network edge environment the same as a Data Centre or Backbone network does not provide the promised results, primarily because of the limited resource availability within the environment. Solutions have to factor in both QoS elements and leverage the resources distributed throughout the network in order to draw benefits from the edge environment. The formulation by Doriguzzi-Corin et al. [41] incorporates latency constraints, but the primary focus of the solution is minimisation of computational costs. Nestor [39] does not provide any QoS constraints and aims to co-locate VNFs onto a minimal number of hosts. The solution presented by Cziva et al. [32] is one of the few solutions that focuses solely on the network edge and prioritises QoS aspects by minimising the overall path latency for all VNFs.

- **Support operator policies**: Placement solutions that are too rigid in their operation are unlikely to provide any benefit beyond academic curiosity, especially for security systems where each operator maintains a proprietary and confidential set of processes and procedures (e.g., firewall VNFs must be hosted on hardened systems) which govern service deployments. Flexibility has tho be supported through integration of operator-defined policies which complement the core placement strategy. To the best of our knowledge, PESS [41] is the only state-of-the-art solution that explicitly supports operator policies during the placement process. However, as these policies can be integrated through additional constraints, formulations by both Dietrich et al. [39] and Cziva et al. [32] can be adapted to conform to the operator best-practices.

Based on this analysis, we argue that there is a distinct need for a solution for the placement,

resource allocation, and traffic steering of security service chains at the network edge. We further propose a model that performs end-to-end Context-Based placement and makes use of residual computational resources at the network edge.

The placement model is formulated as an Integer Linear Programming (ILP) problem to calculate the latency-optimal allocation of service chains in edge networks. Due to their inherent complexity and limited scalability, optimisation problems do not provide adequate operational performance in real-world environments. The placement model outlined below identifies the principles and objectives for the placement of security services within edge networks.

3.6.1 Rationale

Recent advances in virtualisation and NFV allow network functions to be hosted on any physical server, for example edge devices close to the user, servers within the internal infrastructure, or cloud data centres at a distant location.

In order to provide the best user-to-destination end-to-end (E2E) latency, operators are aiming to place VNFs in close proximity to users, on the path that minimises overall transmission delays. To achieve this, edge devices in close proximity to the user are first considered for VNF allocation and falling back to hosting VNFs on devices located within the core network infrastructure (e.g., within a data centre) when the VNF requirements exceed the hosting capabilities of edge devices.

In an ideal scenario, each service chain would be placed on the path with the lowest latency between network endpoints. But implementing such an approach becomes infeasible in real-world deployments. To achieve such behaviour means that every location within the network requires VNF capacity to accommodate requests at any time. In the best-case scenario, where processing capability is not a concern, this strategy requires significant capital expenditure in setting up the required infrastructure. To alleviate this concern utilisation of residual processing capabilities within the network should be employed, while ensuring service-level guarantees. This solution means that the resulting data path through which traffic is steered for services starts deviating from the ideal data path.

In adhering to the Context-Based Security paradigm, the model implements non-sharing of VNFs between multiple devices. As a result of associating a particular security service chains with a single device more flexible security policies are feasible, such as context deviation detection (when a device behaves beyond intended parameters), or mitigation of network attacks

without utilising core infrastructure bandwidth.

The model takes as input a description of the underlying physical network, including the current resource availability of servers alongside one or more security service requests. The output is a mapping of VNFs onto servers within the network, the recommended latency-optimal route between the traffic source and destination traversing the VNF chain, and an updated model that takes into account the resulting allocation.

3.6.2 System Model

We have summarised and synthesised the parameters used for the formulation of the problem and model in Table 3.1. We represent the physical network as an undirected graph $\mathbb{G} = (\mathbb{H}, \mathbb{E}, \mathbb{U})$ where $\mathbb{H}$ is the set of VNF-capable hosts, $\mathbb{E}$ is that of network links between hosts, and $\mathbb{U}$ is the set of users connected to the network.

The heterogeneous nature of the network edge environment implies that the devices present within the network have multiple characteristics (e.g., CPU architecture, CPU clock speed, Memory Architecture, Memory size, underlying OS, etc.). Including all of the characteristics as variables within a problem formulation becomes impracticable. Because of this consideration, we describe the entire system capacity using a single variable. We assume that a VNF can be placed on any host in this graph, and all hosts have a finite hardware capacity (combined *cpu, memory, io, etc.*) to host VNFs, denoted as $W_j, j \in \mathbb{H}$.

Any link within the network is characterised by a latency value A_m expressed in ms, and total link bandwidth B_m expressed in Mbps.

We model flows between two users as (s,d) pairs and a set of paths $\mathbb{P}_{s,d}$ that connect the two hosts. A security service request takes the form $\mathbb{N}_{s,d}$ and consists of multiple VNF $n^i_{s,d}$. Each service request is characterised by the upper latency bound $\theta_{s,d}$ of the associated SLA. This parameter is dictated by the application class associated with a given flow.

We derive latency $l^k_{s,d}$ between the source-destination pairs s,d using path k as the sum of all link latencies. It is worth noting that processing delays incurred by traversal of network functions is not part of this derivation. The complexity in accurate modelling of the processing delays would inevitably create differences between the proposed model and real-world scenarios. A naïve formulation of processing delays assumes an invariant processing latency $\lambda^i_{s,d}$ for any network function $n^i_{s,d}$. The invariant nature would mean that the resulting processing latency for

Table 3.1: System Parameters

Network Parameters	**Description**
$\mathbb{G} = (\mathbb{H}, \mathbb{E}, \mathbb{U})$	Graph of network topology.
$\mathbb{H} = \{h_1, h_2, h_3 ... h_H\}$	VNF Hosts within the network.
$\mathbb{E} = \{e_1, e_2, e_3 ... e_E\}$	All network links.
$\mathbb{F} = \{u_1, u_2, u_3 ... u_U\}$	All flows associated with network functions.
$\Phi = \{(s_1, d_1), (s_2, d_2)(s_3, d_3) ... (s_U, d_U)\}$	All source and destination pairs of flows in the network.
$\mathbb{P}_{s,d} = \{p_1, p_2, p_3 ... p_P\}$	All paths in the network from source s to destination d.
W_j	A singular value that represents cumulative CPU, Memory, I/O, etc. capacity of $h_j \in \mathbb{H}$.
A_m	Latency on link $e_m \in \mathbb{E}$.
B_m	Bandwidth of link $e_m \in \mathbb{E}$.
VNF Parameters	**Description**
$\mathbb{N}_{s,d} = \{n^1_{s,d}, n^2_{s,d} ... n^F_{s,d}\}$	Number of network functions to allocate where $n_i \in \mathbb{N}_{s,d}$ is associated to source and destination pairs $(s,d) \in \Phi$.
$\theta_{s,d}$	Upper latency bound for security service $N_{s,d}$.
$\beta_{s,d}$	Bandwidth requirement for user of security service $N_{s,d}$.
R_i	A singular value that represents cumulative CPU, Memory, I/O, etc. requirements of network function $n^i_{s,d} \in \mathbb{N}_{s,d}$.
Derived Parameters	**Description**
$l^k_{s,d} = \sum_{e_m \in \mathbb{E}} A_m$	Link-level latency between the source-destination pair (s,d) using the path p_k. Derived from the physical topology and the VNF requests.
Variables	**Description**
$X^k_{s,d}$	Binary decision variable denoting if traffic between the (s,d) pair is going through path p_k or not.
Y^j_i	Binary decision variable denoting if network function $n^i_{s,d}$ is hosted on host h_j.
$Z^{ij}_{(s,d)k}$	Binary decision variable denoting if network function $n^i_{s,d}$ is located on host h_j of path p_k between (s,d).

a request is independent of the placement, being only influenced by the VNFs which compose the service chain. A more accurate model relies on the characteristics of the host that serves a given VNF. Thus, processing latency of the form $\lambda^{ij}_{s,d}$ for any network function $n^{i}_{s,d}$ being hosted on server h_j, could be used within the model. However, compilation of such a matrix would lead to an increase of state and lead to an overly constrained model, which could limit the number of requests that can be served. In terms of real-world applicability, it requires that each VNF be profiled on each capable server. This limits the extensibility of a system: adding new VNF hosts requires blocking resources for profiling, and adding servers would be delayed by the need to execute profiling before becoming available for use.

Another issue of contention is the influence that packet I/O has within the VNF. Modern packet I/O solutions focus on providing low-latency from the wire to the processing unit, but require specialised hardware and drivers (e.g., DPDK [48], XDP [2]). Platform-agnostic approaches have higher I/O overhead (e.g., Berkeley Sockets [3], LibPCAP [4]) but work on a wide array of systems. In the scenario where a VNF implementation supports multiple packet I/O methods, each would be considered a different VNF within the model, further increasing the search state and complexity in solving the formulation.

The heterogeneous edge network environments also influence the system requirements of individual VNF implementations. Once again, the inclusion of all possible variables and how they influence the system requirements, becomes infeasible. With similar basis as described for host system capacity, we employ a single resource requirements variable R_i, describing combined the *cpu, memory, io, etc* requirements, for each VNF n_i.

The derivation of variables that are deeply tied with underlying hardware architectures, such as W_j and R_i in our model, has often been of widespread interest and debate within the academic community. While providing a framework for derivation is outside the scope of this thesis, other domains have encountered similar challenges. In Data Centre environments, a solution for this challenge is proposed by Pentelas et al. [108], where a node has a multi-dimensional resource description, is employed. Proving that multi-dimensional solutions are viable in Data Centre infrastructures, where heterogeneity is not an issue, the solution has a non-trivial generalisation for heterogeneous environments. In the domain of exact algorithms and constraint programming, which are related to Integer Linear Programming, as a method for formulating optimisation problems, a solution involving system benchmarks is used. This strategy, frequently encountered when analysing algorithm performance in real-world scenarios, was however challenged

[2] https://www.redhat.com/en/blog/capturing-network-traffic-express-data-path-xdp-environment Retrieved September 2021

[3] http://www.opengroup.org/onlinepubs/9699919799/functions/contents.html Retrieved October 2021

[4] https://www.tcpdump.org Retrieved October 2021

by Prosser [113] with a detailed discussion in § 4.5 and empirical analysis on systems with similar underlying hardware. In a heterogeneous environment, this effect would be present because of the differences in hardware architecture. We however decide to accept this compromise in derivation, looking forward to the evolution of new methodologies.

We use the binary decision variables:

$$X_{s,d}^{k} = \begin{cases} 1 & \text{if the flow } (s,d) \text{ is routed on path } k \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

$$Y_{i}^{j} = \begin{cases} 1 & \text{if VNF } n_{s,d}^{i} \text{ is located on host } h_j \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

$$Z_{(s,d)k}^{ij} = \begin{cases} 1 & \text{if VNF } n_{s,d}^{i} \text{ is located on host } h_j, \\ & \text{belonging to path } p_k \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

The variable described in Equation (3.1) encompasses the path selection and routing decision of flows. Variable (3.2) is linked to the placement decision of individual VNF. We ensure consistency between the two variables using (3.3), which encompasses all of the relevant system parameters.

3.6.3 Problem Formulation

The placement problem is defined as follows:

Problem. *Given the the network $\mathbb{G}$ described by set of users $\mathbb{U}$, the set of VNF Hosts $\mathbb{H}$, the set of network edges $\mathbb{E}$, with the traffic flow pairs Φ, the set of VNF chain requests $\mathbb{N}_{(s,d)}$, and a latency matrix l, we need to find an appropriate routing of all flows and placement for all VNFs that minimises the total expected end-to-end latency from all users to their destinations.*

The solution to the placement problem is given by:

$$min. \sum_{(s,d)\in\Phi} \sum_{p_k\in\mathbb{P}_{s,d}} X^k_{s,d} l^k_{s,d} \tag{3.4}$$

Equation 3.4 looks for the values of $X^k_{s,d}$, while it is subject to the following constraints:

$$\sum_{p_k\in\mathbb{P}_{s,d}} X^k_{s,d} = 1, \forall (s,d) \in \Phi \tag{3.5}$$

Constraint (3.5) ensures that only one valid path may be used for any given flow. This means that all network traffic belonging to a given flow follows the same network path, which contains VNF allocations on physical servers. The constraint gives two guarantees: that there are no instances of VNFs performing the same task, and that the network traffic traverses all VNFs on the Service Chain.

$$\sum_{p_k\in\mathbb{P}_{s,d}} X^k_{s,d} l^k_{s,d} < \theta_{s,d}, \forall (s,d) \in \Phi \tag{3.6}$$

Constraint (3.6) verifies that the selected end-to-end (E2E) path latency is below the upper bound specified for the flow at the time of placement ($\theta_{s,d}$). This means that the application traffic traversing the selected path and associated VNF allocations will be below the tolerated latency threshold for the application used. The calculation for all possible latency values is stored in the latency matrix $l^k_{s,d}$ where the s,d pair indicates the traffic source and destination and k denotes the network path used between the two points.

$$\sum_{p_k\in\mathbb{P}_{s,d}} X^k_{s,d} l^k_{s,d} + \sum_{n^i_{s,d}\in\mathbb{N}_{s,d}} \sum_{h\in\mathbb{H}} Y^j_{(s,d)i} \lambda^{ij}_{s,d} < \theta_{s,d}, \forall (s,d) \in \Phi \tag{3.7}$$

If a requirement to incorporate processing latencies within the path latency selection is present, an extended formulation taking the form expressed in Constraint (3.7) can be used instead. However, in further considering this formulation, we refer to the formulation only containing link latency.

$$\sum_{(s,d)\in\Phi}\sum_{p_k\in\mathbb{P}_{s,d}} X^k_{s,d}\beta_{s,d} < B_m, \forall e_m \in \mathbb{E} \tag{3.8}$$

Constraint (3.8) verifies that links do not become overloaded by the path selection. It takes the bandwidth requirements of all flows traversing through the link and ensures that their sum does not exceed the maximum link bandwidth.

$$\sum_{h_j\in\mathbb{H}} Y^j_{(s,d)i} = 1, \forall(s,d) \in \Phi, \forall n^i_{s,d} \in \mathbb{N}_{s,d} \tag{3.9}$$

Constraint (3.9) states that each VNF must be allocated to exactly one host. This is done by verifying that the allocation counter $Y^j_{(s,d)i}$ for a particular VNF sums up to 1.

$$\sum_{(s,d)in\Phi}\sum_{n_i\in\mathbb{N}_{s,d}} Y^j_i R_i < W_j, \forall h_j \in \mathbb{H} \tag{3.10}$$

Constraint (3.10) enforces that resource limitations of hosts are adhered to (CPU, memory, IO are finite and can only support a limited number of VNFs on any given host). This constraint sums up all of the resource usage requirements R_i (selected by the binary decision variable Y^j_i) and ensures that for all hosts h_j the individual sums are below W_j, the total capacity of the host.

$$Z^{ij}_{(s,d)k} = X^k_{s,d} Y^j_i \tag{3.11a}$$

$$Z^{ij}_{(s,d)k} \leq X^k_{s,d} \tag{3.11b}$$

$$Z^{ij}_{(s,d)k} \leq Y^j_i \tag{3.11c}$$

$$Z^{ij}_{(s,d)k} \geq X^k_{s,d} + Y^j_i - 1 \tag{3.11d}$$

Constraint (3.11a) is used to guarantee that the allocation is valid with respect to the path. This is

done by verifying that the chosen host for a VNF allocation is located on the selected path k between (s,d). The linearised form of Constraint (3.11a) can be expressed by Constraints (3.11b), (3.11c), (3.11d).

$$\sum_{p_k \in \mathbb{P}_{s,d}} Z^{ij}_{(s,d)k} >= 1, \forall (s,d) \in \Phi, \forall n^i_{s,d} \in \mathbb{N}_{s,d}, \forall h_j \in \mathbb{H} \tag{3.12}$$

Finally, we enforce the chaining requirement through constraint (3.12), where all VNF belonging to the same Service Chain are on the same path. Doing so ensures that a chain does not reside on multiple paths, violating network traffic steering considerations by e.g., duplicating packets.

3.7 Summary

The design concepts and core requirements for a security-oriented NFV framework capable of creating, running and dynamically managing virtual network functions in various emerging network environments were presented in Section 3.2. Section 3.3 discussed the relevant considerations in designing such a framework.

Section 3.4 outlined the need for using customisable microservice network security modules in the proposed networks as opposed to monolithic software architectures and bespoke implementations of lightweight security functionality, providing much lower hardware requirements, and allowing for rapid creation and deployment of services suited to the operational environment.

Finally, Section 3.6 presented a solution for the latency-optimal placement of VNF chains and traffic steering to achieve low-latency security service deployment and dynamic management. A formulation of an optimisation problem using Integer Linear Programming (ILP) is given, with the associated system model and description of the individual elements.

Overall, the design principles presented in this chapter will allow network operators to deploy customisable security systems in edge networks and manage the placement of services to provide the best latency for their users.

Chapter 4

Implementation

4.1 Overview

Following the design considerations outlined in Chapter 3, a network security function orchestration framework has been implemented.

The framework has the following main characteristics:

- **Context-Based Network Function Control**: Network Functions operate within the context of the network users they provide services for. In order to cater to different needs, from both an end-user perspective and network operator requirements, a robust control plane is presented. The control plane implements the principles of context-based network security in service request handling, lifecycle management, and identification of new requirements.
- **Real-time Allocation of Services**: Whenever a new device joins the network, or new applications are used, the associated services must be allocated onto VNF hosts, provisioned and activated. In order to offer enhanced user Quality of Experience, these new services are made available in real-time, with a user experiencing minimal delays from instantiating new VNFs. For example, Cziva et al. [37] show that 50 container-based VNFs can be created and started in 10 seconds, but does not account for orchestration overheads. The orchestration must also be performed in a similar timeframe (e.g., device detection, VNF placement, container transfer, etc.), with a goal of 20 seconds for VNF availability.
- **Lightweight Network Functions for Edge Environments**: Modern applications require

minimal network latency. Packet processing overheads of network functions have to be kept to a minimum to meet these demands. Microservice-based network functions that perform lightweight, targeted processing are employed to achieve the proposed goal.

The architecture is organised into four planes, based on the recommendations of the ETSI NFV standard [42]:

- **The Infrastructure plane** consists of edge devices and network infrastructure. Lifecycle management of individual network functions is delegated to the Infrastructure plane in order to minimise management overhead and enhance infrastructure resilience.
- **The Virtual Infrastructure plane** manages the configuration of network traffic routes, detects new clients and applications used, communicates with edge devices used for VNF hosting, and provides per-host utilisation metrics used in orchestration.
- **The Orchestration plane** handles service request prioritisation, mapping of operator policies to service function chains, placement of individual services, traffic steering to provide latency-optimal utilisation, and tracks infrastructure availability.
- **The Service plane** consists of individual network functions that can be used to provide necessary functionality resulting from the requests for network services.

The following sections present the implementation details of the components of the framework. The network architecture, based on real-world topologies and influenced by existing principles of Edge network designs, is presented in Section 4.2. In Section 4.3 the implementation of the control plane aspect will be detailed, with focus on service request management and identification of new application requirements. Lifecycle management of network functions and deployment of new services within the network are described in Section 4.4. We continue with a heuristic solution aimed at providing real-time allocation of service requests in Section 4.5. Section 4.6 gives insight on implementation and use of lightweight, microservice-based security services. Finally, we conclude the chapter in Section 4.7.

4.2 Network Infrastructure

The infrastructure of the network consists of various resource-constrained hosting platforms for network functions, such as Single-Board Computers, home routers, and lightweight NFV

servers, co-existing with public cloud VMs for computationally-intensive tasks. These types of devices are susceptible to "command-and-control" attacks[1], where a single device issues a network-wide instruction to perform undesired or malicious activities (e.g., DDoS attacks or data theft) that compromise the entire infrastructure.

The underlying network interconnecting these devices uses an SDN-compliant OpenFlow design for routing, enabling fine-grained control of the network flows from applications. As a result, OpenFlow-compatible switches are present throughout the network to allow centralised control and installation of forwarding rules on all devices required for VNF traffic steering. Clients using the network are detected by the SDN Controller through switch notifications, in order to identify the security context in which they are operating. Specifically, the OpenFlow 1.3 protocol is used to control all network elements without using vendor-specific interfaces or extensions and communicating device presence to the Security NFV framework.

Management of the SDN infrastructure is through networks designed for sparse latency-sensitive traffic where throughput is generally of little concern compared to the data-plane network. The network is thus designed for reliable and consistent performance. The potential implementations include the following alternate management network strategies:

1. **In-band:** The management and data networks are shared, with no isolation between the two traffic types. Management traffic is subject to congestion and reduced bandwidth arising from data traffic.

2. **Logically Out-Of-Band (OOB):** Management traffic is logically separated, with approaches using, e.g., VLANs or dedicated flow rules in switches. This approach allows enforcement of QoS by prioritising management traffic. Isolation is enforced at QoS-enabled routers, with no complete guarantees of separation between two network locations.

3. **Physically OOB:** A dedicated network is used exclusively for management traffic. While the approach involves significant investment for new switches and host network interfaces, it is used in critical environments. Physically separated networks are the recommended solution for OpenStack cloud software used for NFV frameworks, and employed in many production cloud Data Centre environments.

In our implementation the control network uses an in-band solution with no isolation. The simplicity of the solution allows for rapid development, deployment, and evaluation of the proposed framework.

[1] https://www.paloaltonetworks.com/cyberpedia/command-and-control-explained Retrieved August 2021

In terms of OpenFlow applications used, which are essential for the operation of the network, we employ a few OSI Layer 2 (Ethernet) and Layer 3 (IP) applications to enable basic network functionality. Because the OpenFlow switches, by default, do not perform any network forwarding not present within the forwarding tables, the entire behaviour of network switching and routing protocols have to be re-implemented as OpenFlow controller applications. The lack of a Link-Layer Discovery Protocol [65] support for the switches makes discovery of new devices (e.g., clients joining the network) increasingly difficult. The Address Resolution Protocol (ARP) [112], used by network clients to map MAC addresses to IP addresses and issue valid Ethernet packets, is also not supported out-of-the-box.

Industrial solutions acknowledge this limitation of the OpenFlow specification. Starting with OpenFlow Switch specification 1.1 [2], a hybrid pipeline processing mode is defined, which supports both OpenFlow operation and normal Ethernet switching through a classification mechanism that routes traffic to either pipeline. However, the standard considers this classification outside the scope of the standard, which led to vendors defining their own implementations. For example, nVidia Mellanox Onyx performs classification based on VLAN filters [3], Nokia SR OS supports per-interface or per-VLAN OpenFlow pipelines [4], and Arista EOS performs either per-interface or per-VLAN classification, but can also perform a subset of OpenFlow actions for all packets [5]. If the vendor-prevalent VLAN classifier is used in a OpenFlow hybrid network then the host needs to be aware of the underlying network configuration (e.g., knowing the OpenFlow pipeline VLAN ID) and segregate network behaviour based on network characteristics; from a security perspective, this poses a risk by circumventing the OpenFlow pipeline (and, implicitly, the security service chain that the operator enforces).

4.3 Network Control Plane Module

Alongside applications implementing standards-defined network protocols, we make use of a dedicated Context-Based Control Plane Module that connects the OpenFlow infrastructure to the NFV platform. The functionality of this module is detailed within this Section.

The Control Plane Module is an SDN controller module that monitors for new application flows from end-users. The module is built for the Ryu SDN controller for our prototype. It features a

[2] https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.1.0.pdf Retrieved August 2021

[3] https://docs.nvidia.com/networking/display/ONYXv381112/OpenFlow+1.3+Workflow

[4] https://documentation.nokia.com/html/0_add-h-f/93-0073-HTML/7750_SR_OS_Router_Configuration_Guide/openflow Retrieved August 2021

[5] https://www.arista.com/en/um-eos/eos-openflow Retrieved August 2021

southbound API (Application Programming Interface) that communicates with the NFV Orchestrator component for lifecycle and function management, by transmitting information regarding device type and network behaviour.

The controller applications are responsible for guaranteeing the network availability of new devices and applications. The LLDP application monitors device links by periodically sending an OpenFlow PacketOut message with Ethernet Packets containing LLDP data on all ports of a switch. Compliant network devices are also expected to send LLDP messages which are forwarded by the receiving switch to the Controller. The LLDP controller application is thus aware of all of the neighbours of a switch, including client devices which join the network.

We extend this mechanism to allow for new application presence and inform the NFV framework for the deployment of security modules. Detection is made by the OpenFlow receiving Switch OpenFlow PacketIn notifications for new application flows, which do not have table entries setup. This detection triggers the VNF Chain allocation process for network chains, with a high-level overview of the process presented in Figure 4.1.

We use a dedicated **Device Manager** that, based on a device's properties and expected behaviour (e.g., smart phone, IoT sensor, laptop, etc., and social media, video streaming, web browsing, etc., respectively), gathered from LLDP information and application flow details, requests the associated **Contexts** which describe the cybersecurity requirements related to network behaviour. The **Context Manager** is responsible for the composition of the security service function chain that respects the operator's best practices and requirements with respect to the aforementioned behaviour.

In accordance to the principle of context-based security, the NFV framework maps per-device application traffic to unique VNF chains, by handling identification of appropriate VNFs, service chain composition, traffic steering, and VNF orchestration. The mapping and chain composition is defined by the network operator according to their best practices and function availability: high-level policies that can be inferred from the network traffic (e.g., HTTPS, VoIP, etc.) are associated with series of network functions descriptors with security functionality. A NF descriptor in our framework includes the archetype of the network function (e.g., generic firewall, IDS, DPI, etc.), along with specific configuration requirements that have to be applied to the network function (e.g., IP address, TCP or UDP ports, match rules, etc.).

The chain routing is mutually exclusive with ECMP (Equal-Cost Multi-Path) routing techniques, as a chain does not allow traffic merging from multiple flows. This implementation choice aligns with the design decision for adoption of Context-Based Network Security in Section 3.2 and is

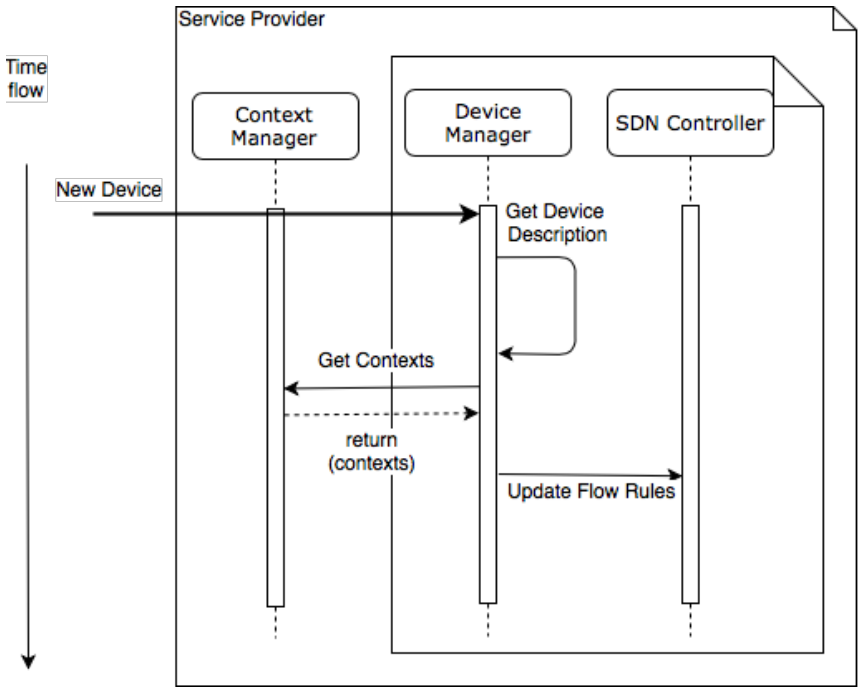

Figure 4.1: High-level behaviour of device identification and SDN routing

offset by the choice of lightweight network functions, expanded on in Section 4.6. The SDN controller module, given a series of routing paths generated after the allocation of service function chains by the NFV Orchestrator, installs the relevant forwarding table entries on switches, and verifies existing functionality of traffic steering.

The southbound API is used for VNF lifecycle management, when allocated service chains are not in use. Under these conditions, the NFV Orchestrator notifies the controller module of the deallocation decision by monitoring LLDP disappearances. The network switches involved in traffic steering are instructed to drop the forwarding table entries pertaining to the disconnected device.

Finally, this module can be extended with northbound API capability to give a visual User Interface for network engineers to view and manage data in real-time. One way to achieve this is to provide a REST (REpresentational State Transfer) API, alongside a web application built using HTML5, CSS3 and JavaScript components. The API primitives included are related to VNF categories, management of service chains and policy mapping, specifying network-wide configuration parameters and inspection of the network infrastructure state.

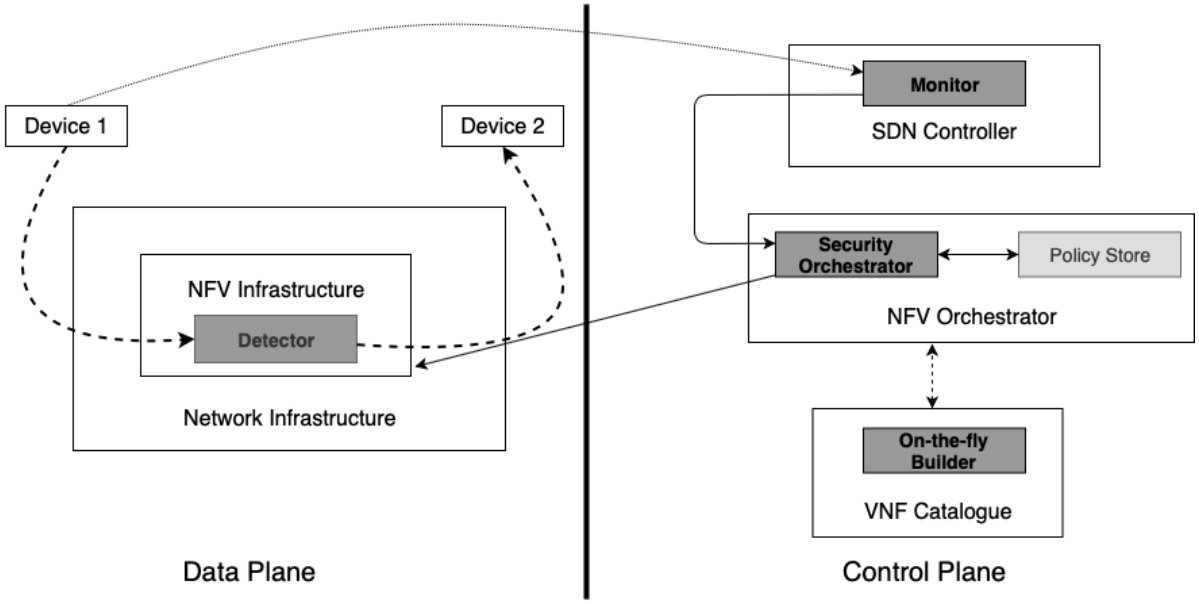

Figure 4.2: Components for Security VNF Management

4.4 Network Function Management

Security function management, including orchestration, individual function lifecycle management, function building, and VNF host management is performed using components derived from the ETSI NFV reference architecture [42]. A high-level overview of the system is provided in Figure 4.2. The salient components are: **the function builder**, **the VNF host monitor**, and **the function orchestrator**. For the basis of our framework, we have used Kubernetes, an open-source, production-grade container orchestration mechanism. The remainder of this Section presents implementation details related to the security-centric NFV functionality.

4.4.1 NFV Orchestrator

The NFV Orchestrator responsible for interacting with the underlying network infrastructure through the SDN Controller, for management of network function creation, deletion, state management, and access, and for management of the underlying NFV infrastructure: communicating with servers capable of hosting NFVs, tracking global and local resource availability, storage systems, and distributing operational data throughout the cluster.

In the traditional Kubernetes architecture, the Orchestrator is comprised of several independent modules (e.g., scheduler, data store, controller manager), collectively called the Master Node,

interconnected by the Kubernetes API Server[6]. The Kubernetes **scheduler** is responsible for mapping containers on capable worker nodes, the **controller manager** is responsible for monitoring servers joining the cluster and watches for tasks that need to be handled (e.g., creation of Kubernetes pods, setting up routes), and the data store is responsible for maintaining the state of the cluster as a key-value store. Our implementation extends the **scheduler** module and **controller manager** components. It also exposes the Kubernetes API to the SDN controller, to monitor for service requests.

The Kubernetes nodes (equivalent to VNF Hosts in the ETSI NFV reference architecture) contains several components as well. The **kubelet** agent monitors the state of each node, handles container management, and informs the Controller of it's state. **kube-proxy** is responsible for setting up the node's network rules for routing traffic between pods and the wider network, exposing services and network ports. In operation, it makes use of the operating system packet filtering functionality. Finally, the **container runtime** is responsible for the actual container execution, with support for implementations making use of the Container Runtime Interface specification[7].

The orchestrator, upon receiving an event notification from the SDN controller module, is responsible for identifying the required security policies related to the device, generating a placement decision for the network functions of the associated chain, requesting the creation of microservice security artifacts, and generating a steering path for application traffic. It is responsible for the migration of a network security service (or suite of services) in the event of roaming devices and reconfiguration of deployed functions to react to evolving network conditions.

Most notably, we defer some aspects of lifecycle management of network functions (e.g., shutdown) to VNF Hosts to minimise the control traffic, because of the filtering nature of security functions. The orchestrator receives regular updates from hosts within the infrastructure to determine placement availability and liveliness.

In order to provide online placement of network functions in dynamic networks, alongside the Inline Service Chaining model proposed in Section 3.6, we have also implemented a heuristic algorithm detailed in Section 4.5.

[6]https://kubernetes.io/docs/concepts/overview/components/ Retrieved November 2021

[7]https://github.com/kubernetes/community/blob/c23c61872559400107e0863600ecf898841299ea/contributors/devel/sig-node/container-runtime-interface.md Retrieved November 2021

4.4.2 Network Function Building

The Kubernetes architecture uses an external container registry for storing pods. The default registry is the Docker Hub[8], with default behaviour for Kubernetes nodes to download (via the Internet) pods that are not available on the node. Advanced cluster setups, and cloud-based clusters (e.g., Google Kubernetes Engine, Amazon Elastic Kubernetes Service), can be configured to use external registries.

In order to achieve lightweight network functions with minimal overhead that are available in a heterogeneous environment, we use an on-the-fly VNF building process. Located within the VNF Catalogue module of the ETSI NFV architecture [42], of Kubernetes Image Registry, this module produces software artifacts for the target hardware and intended functionality.

From a systems implementation perspective, there are several approaches to achieving artifact generation, presented below:

- **Static embedding:** The bespoke artifact contains the targeted detection instructions embedded into it. This provides rapid deployment and activation times, with minimal system requirement overheads. Altering the behaviour involves generating, transferring and deploying a new artifact, which leads to increased downtime.
- **Dynamic linking:** The bespoke artifact is a shared library with a well-defined interface for interacting with a generic VNF module. Using jump tables and relocation pointers, this adds a small memory footprint overhead and increases processing overhead. Changes in detection requires minimal downtime, as the only change required is updating of jump tables and relocation pointers. The reliance on dynamically linked external libraries limits operation in heterogeneous environments due to source code portability and assumptions made during implementation regarding the underlying software components.
- **Run-time configuration:** A VNF can be configured at run-time to perform network functionality using human-readable configuration files. The configuration is read and processed into data structure representations that can be used for packet processing. The overhead incurred is larger, as data structures (e.g., tries for DPI) need additional memory and there is dedicated logic to interpret the information.

The simplicity of embedding targeted packet processing logic into application binaries, with the performance benefits compared to the other approaches are the main driving force for use of

[8]https://hub.docker.com Retrieved November 2021

static embedding in our framework.

Our on-the-fly builder module receives a request from the orchestrator to use a VNF archetype with a given specialisation (e.g., firewall; block TCP/80 in the case of HTTPS applications) and a desired VNF host to act as a destination. To achieve static embedding, we give the compiler preprocessor directives to selectively enable codepaths and indicate traffic patterns to match against. The builder generates the software artifact to be used in the given situation and uploads it to the VNF host for deployment.

The resulting container image is built on top of Alpine Linux, chosen for its lightweight nature and support for multiple hardware architecture. The final container image has the embedded software artifact and execution entrypoint set to it, resulting in a lightweight image (in the order of a few kilobytes) that can be transferred to the VNF host.The associated deployment is then created using the resulting container image.

4.4.3 VNF Host Management

The VNF Hosts located within the infrastructure contain a monitor module that reports liveliness and utilisation to the NFV Orchestrator. The liveliness and utilisation information is used in discovery of new VNF Hosts by the orchestrator, and can be used to determine available capacity for VNF instances. In the Kubernetes model, this functionality is performed by the **kubelet** agent.

In the event of a fault related to a given host, the orchestrator can recalculate the placement of the VNFs deployed to that host. Implementation of this fault tolerance mechanism allows for increased resilience of the overall network infrastructure, through timely detection and mitigation of independent failures.

Furthermore, Kubernetes can also detect individual failed VNFs, through pod monitoring functionality. We periodically monitor individual pod status, to both identify critical bugs within the network function itself, unexpected resource contention, or malicious attacks specifically targeting the security infrastructure.

4.4.4 Function Lifecycle Management

Because security function chains filter undesired traffic, functions at the end of a service chain might become idle. To enhance resource utilisation, we propose the offloading of certain lifecycle management aspects to VNF Hosts. For the shutdown of inactive network functions, we use the monitoring capabilities of the function hosts.

In Kubernetes, the closest equivalent workload would be a Job, which creates and runs one or more containers (and retries execution) until a specified number successfully complete execution. This requires network functions to terminate after a period of inactivity, requiring specific functionality that limits general applicability. Instead, we leverage the node capabilities, namely the **kubelet** agent, which already monitors container state. We extend the agent with monitoring of network traffic statistics (e.g., under Linux we can monitor an interface's packet counters using the `/sys/class/net/net1/statistics/rx_packets` kernel interface) for a given pod. This mechanism provides low management overhead, as the agent already periodically monitors pod status.

If a network function does not receive any network traffic within a given temporal window, a host can notify the orchestrator of the inactivity and perform shutdown of the inactive function. We reason that the situation occurs under two circumstances, presented below:

1. **Application stopped:** As there are no standardised network mechanisms for announcing the end of a user application's lifetime (e.g., device left the network), we use flow inactivity to infer that an application is not actively used by the end-user.
2. **Previously filtered traffic:** Undesired network traffic is filtered earlier in the service chain. In the scope of Context-Based security, this would require deployment and use of new, more aggressive security policies.

In either of the above-mentioned situations, proactive reclaiming of resources from inactivity allows enhanced flexibility in catering to multiple users' needs in dynamic environments.

We also extend **kubelet** to gather information on VNF alerts issued by the network function (by monitoring the function's standard error output), the VNF status (as described above), and the operational state within deployment. Each Kubernetes node then sends a liveliness report to the master, to ensure state synchronisation within the cluster and give an update on the pod availability. On the Orchestrator side, the data store would be updated with the information on individual VNF status, liveliness, and any alerts raised. In conformity with the Kubernetes

Code Listing 1 Heartbeat Message Structure

```
"conditions": [
    {
        "vnfid": "Mirceas-iPhone-0",
        "kind": "Ready",
        "status": "True",
        "vnfalerts": ""
    },
    {
        "vnfid": "Mirceas-iPhone-1",
        "kind": "Ready",
        "status": "True",
        "vnfalerts": "[FW]: IP;192.168.0.1"
    },
    {
        "vnfid": "Mirceas-iPhone-2",
        "kind": "Ready",
        "status": "Idle",
        "vnfalerts": ""
    }
]
```

data model, these updates would be further propagated to other modules. For example, VNF Alerts would be forwarded to a Network Security Monitor which presents a centralised view of potential incidents to an operator.

An example of the liveliness report combined with VNF lifecycle information is given, in JSON format, in Code Listing 1. In this particular scenario, the VNF *Mirceas-iPhone-1* produces an alert for a dropped packet. As a result, the following Network Function, *Mirceas-iPhone-2* reports an *Idle* status as it received no network traffic. If the behaviour persists over several messages, the Kubernetes node would prompt the automatic termination of the *Mirceas-iPhone-2* VNF, excluding it from the heartbeat message and informing the controller of the freed up resources.

4.5 Heuristic Minimal Path Deviation

The Inline Service Chaining model proposed in Section 3.6 is designed to provide an optimal placement for security service chain requests within a given network topology. Given the complexity of the ILP model, the time required to produce solutions is outside the acceptable range

Code Listing 2 Minimal Path Deviation Algorithm

```
def mpda_allocation(flows):
        for f in flows:
                chain = get_chain(f)
                paths = get_paths(f.src, f.dst)
                paths.sort(key=len)
                for p in paths:
                        if can_allocate(chain, p):
                                a = allocate(chain, p)
                                update_resources(a)
                                allocations.add(a)
```

Code Listing 3 Evaluation for chain allocation

```
def can_allocate(chain, path):
        remaining_vss = chain.copy()
        for vss in chain:
                remaining_vss.pop()
                for server in path:
                        if not can_host(server, vss):
                                continue
                        vss.potential_host = server
                        break
        if remaining_vss:
                return False
        return True
```

for the dynamic scenarios the framework caters to. Furthermore, the model assumes a coherent and consistent snapshot of the network-level information (e.g., link-layer latencies) that are subject to temporal variations. As a result, we implement a heuristic algorithm to find near-optimal solutions rapidly, while still conforming to the context-based security paradigm by maintaining VNF separation for multiple devices.

Obtaining an instantaneous network state with the required properties (e.g., link latencies) is difficult to achieve in large, multi-link edge networks. For general network edge environments, the distances between network nodes range between a few meters (in the case of dense small-cell networks: e.g., 5G, WiFi) to a few kilometres (for customer-provided equipment connected to a demarcation point). The short distance results in reduced propagation delays and can be leveraged for reduced latency in service placement. Exchanging in-network measurements with network distances is a reasonable trade-off in placement of services in realistic edge environments.

Consequently, we propose a Minimal Path Deviation Allocation heuristic that conforms to realistic network conditions. Our algorithm performs incremental allocation of service chain requests based on deviation from the shortest network path. Our allocation strategy still follows the same high-level design choices as the optimal solution presented in Section 3.6, namely inline network function chains, with each VNF serving a single user, and considerations of temporal variances in the network infrastructure, but relaxes the optimality requirement with respect to measured link-layer latency.

In the Kubernetes model, pod allocation and placement occurs within the **kube-scheduler** component. The default behaviour is performed in two steps: *filtering* and *scoring*. Filtering finds the set of Kubernetes nodes which have the capability of running a pod. Scoring, as the name implies, scores the nodes to choose the most suitable host for the pod. Some of the scoring features that can be used by the scheduler refer to image locality (favouring nodes that have the container image preloaded), node affinity (placement on nodes with certain operator-defined labels), pod topology location (favouring nodes within a certain region), resource availability and balancing (favouring nodes where resource availability has certain criteria, e.g., most available, most used, or balanced usage).

Because the default Kubernetes behaviour has limited knowledge on the network topology aside from the use of zones, and does not provide a simple mechanism for node adjacency preference, we have implemented an alternative scheduler module in Python. This module gathers network topology information from the SDN controller and uses it in conjunction with the Kubernetes node capability information. Scheduling is performed on the entire service chain, defined within the Kubernetes model as a Deployment involving multiple pods.

The implementation makes use of the Kubernetes Python API client[9]. For the sake of brevity and to restrict the presentation of the work to novel elements we choose to include in this thesis the high-level pseudocode equivalent of the implementation. The pseudocode for the approach is listed in Code Listings 2 and 3, expressed using Python-like syntax.

Similar to the Kubernetes scheduler, we divide the placement process in two distinct steps: path identification and node selection. In Code Listing 2, we consider all possible paths for allocation, and sort them based on path length. For the proof-of-concept implementation, all possible simple (i.e., non-cyclic) paths between the flow source and destination are considered. The path search algorithm is based on the algorithm proposed by Yen [153], which provides an ordered list of paths between the source and destination, starting from the shortest. In terms of algorithmic complexity finding the first K paths requires $O(KV^3)$ operations, with V being the

[9]https://github.com/kubernetes-client/python

order of the graph (or, in our case, the number of VNF hosts). An alternative option within the scheduler, for graphs with a small number of nodes (e.g., under 5, where convergence occurs), a modified breadth-first search [126] can be used, as the number of operations is $O(V!)$ for obtaining the same result. The result of the path search algorithm is a generator, which lazily computes the next result of the path search, avoiding the need to pre-compute all network paths, as $K \rightarrow V!$.

Analysing the paths in order, we determine if a service chain can be allocated on the given path using the outline presented in Code Listing 3. For any given path and service chain, we consider placement of each constituent VNF within the chain on the closest server. For a basic implementation, the `can_host` method only analyses the resource availability for the server and compares it to the requirements of the VNF. If successful, the scheduler marks the function instance for allocation onto the server the resource requirements are met (e.g., number of hosted pods is less than the server capacity). If the service chain can be fully placed on a given network path, the allocation is performed and the model of network resource availability is updated accordingly.

In terms of operator customisation, through implementation of best practices, we take inspiration from the Kubernetes `NodeAffinity` policy: pods associated with special conditions contain a special key-value map, called a `nodeSelector` which specifies certain requirements for the pod. In scheduling a service chain, when considering a VNF with a specified `NodeAffinity` key-value map for allocation we filter nodes on the path under consideration. As an example, if an operator deems that IDS VNFs are too computationally expensive to be run outside of commodity servers, it can specify `nodeSelector: {"node-type": "commodity-server"}` for VNF Hosts. For IDS pods, within the pod specification, the `NodeAffinity: {"node-type": "commodity-server"}` should also be present. The scheduler policy would only place the resulting VNF onto supported hosts.

4.6 Example Network Functions

This section details the implementation of microservice security-focused network functions which are commonly encountered in provider networks and the associated on-the-fly build system used to generate the VNF instances dynamically based on individual client requirements. We introduce the generic *wire* network function, then construct two commonly encountered security VNFs: a *firewall* and a *Deep Packet Inspector*.

4.6.1 Overview and Common Components

The implementation[10] of the functions is done in C and is designed for minimal dependency on external libraries. The only concrete dependency of the framework is the C Standard Library, for interaction with standard streams and argument parsing.

The C programming language was chosen because it is the de-facto system programming language, offering high performance and nearly complete control over memory allocation and access. The majority of the packet I/O libraries are based on, or offer support for, C, making extensions to the general framework approachable. Alternative languages and frameworks for implementation have been considered. Rust [77] is also designed for performance and memory safety, but the language specification is not yet, in our opinion, stable (i.e., the language underwent several revisions in the 2015-2021 timeframe). The Click framework [78] allows rapid composition of network functions, but operation in heterogeneous environments requires porting of individual elements due to memory alignment issues and the framework overall is still considered experimental.

The main abstractions of the framework are contained within a C header, `uNF.h`. The header contains abstractions and functionality for network interface acquisition and release, as well as code which performs packet I/O. Through the API, network interfaces are accessed by their name (e.g., `eth0`, `net1`, `enp0s3`, etc.), and the internal data structure makes use of a linked list to store the associated file descriptors acquired from the I/O library. Because a network function generally uses 2-3 interfaces (ingress, egress, and the possibility of a tap interface), the size of the internal data structure is small, and the string-based public API can be used by developers in gathering debug information without exposing details on the platform-dependent implementation (e.g., whether the file descriptor is POSIX-compliant or a pointer towards a library data structure).

It is be the only code location which has awareness of the specific functionality of the underlying platform (e.g., packet I/O method, interface acquisition, OS implementation details), in order to isolate platform-dependent code and provide a single location for the extension of the platform. For example, the Berkeley Socket platform support consists of 200 lines of C code, with the majority of the lines being boilerplate socket acquisition and binding. The socket is set up to receive entire Layer 2 packets: `AF_PACKET` family with `ETH_P_ALL` protocol to receive all protocols.

Although the intention of the framework is to avoid the use of heap memory, as it could lead

[10] https://github.com/mirceaIordache/uNF

to unsafe code being developed and would lead to unpredictable memory usage at runtime, the platform-dependent code is exempt from this limitation. For example, listing all of the network interfaces using the Berkeley Socket interface is done through the `getifaddrs` function which performs heap memory allocation and requires a subsequent call to `freeifaddrs` for freeing the heap memory.

4.6.2 On-the-fly Building of VNFs

Based on the considerations outlined in Section 3.4, the resulting lightweight VNFs should use minimal runtime configuration, in order to provide predictable activation times and resource usage. To this end, the software artifacts are built upon request, after a suitable hosting server is identified.

The generation of VNFs is done by issuing parametrised requests for compilation to a build server. The compilation process produces a binary program which contains the security-related functionality built-in, with little dependence on dynamic allocation of resources (e.g., heap memory).

In the prototype implementation of the lightweight VNF build system, the parameters presented in Table 4.1 have been identified for use in the generation of the artifacts. The build system is based on the GNU Compiler Collection suite, and the GNU Make utility for configuration of the different parameters requested. The **ARCH** parameter dictates the target architecture of the server that is to host the VNF, and is used for ensuring the correct processor instruction set is employed in the final artifact. The **PKT_CAP** variable is required to use the network packet acquisition library best suited for the environment; multiple similar libraries, with distinct advantages and subsequent limitations are available (e.g. Snort DAQ, Intel DPDK, Linux/BSD Sockets, etc). For Intrusion Detection Systems, the **IDS_RULES** are provided as a list of known signatures that are to be transformed at compile-time into representations of the data structures used by the DPI engines, in this case a non-wildcard version of Aho-Corasick tries. Finally, the **FW_RULES** parameter may be used to define any firewall-related rules that involve filtering of traffic.

The parameter list, as presented in Table 4.1, is non-exhaustive in order to maintain the flexibility of security services. It can be further extended to include other types of security functionality. As an example, for Stateful Load Balancer VNFs, which are sometimes used as DDoS mitigation techniques, one extension includes the number of simultaneous connections that one server can have at any time through a **LB_CONN** parameter extension.

Parameter	Values	Description
ARCH	x86, armv7, mips	Target platform hardware architecture (**Required**)
PKT_CAP	dpdk, daq, bsd_socket	Packet acquisition library (**Optional - default bsd_socket**)
IDS_RULES	OUT:TCP;80;"GET" IN:TCP;21;"root"	DPI signatures to be matched against (**Optional - default none**)
FW_RULES	IP;192.168.0.1	Firewall DROP rules to be used (**Optional - default none**)

Table 4.1: Build system parameters, possible values, and explanations

Code Listing 4 Basic Wire NF implementation

```
1  int main(int argc, char** argv) {
2      //...
3      unf_listen(ingress_intf);
4      unf_listen(egress_intf);
5      do {
6          unf_recv(ingress_intf, packet_buf);
7          //Wire NF. Pass data along
8          unf_send(packet_buf, egress_intf);
9      }
10     while (1);
11 }
```

4.6.3 The Wire VNF

The simplest form of VNF that can be developed, and from which other implementations are derived is a simple *wire* function that forwards packets from an ingress interface to an egress.

The implementation relies on few lines of C code, the summary of which is presented in Code Listing 4. The complete implementation, including pre-processor directives (e.g. header `#includes`) consists of 50 lines of source code. The presented Listing has the following behaviour: on Lines 3 and 4 the ingress and egress interfaces are set up in userspace. The result is stored within the internal data structure of the uNF framework, the application developer being encouraged to use the canonical interface names in operation.

After interface acquisition, the program executes an infinite loop (Lines 5-10) through which a network packet from the ingress interface is received in a buffer, and is sent on the egress buffer. The behaviour of the memory manipulation mechanisms are dictated by packet I/O interface: Berkeley Sockets copies the data into the buffer, while DPDK can use zero-copy mechanisms. The packet buffer size is set to the network MTU, which for networks including Jumbo frames

is of size 9234 bytes: The Ethernet header is (up to) 18 bytes, IP header is 20 bytes, with 9214 bytes for subsequent layers (IP Payload), as per upper limits of common switch vendors[11][12].

Although not pictured, or required (the lifetime of the container is tied to the lifetime of the network function, meaning that the resources of the container would be reclaimed when the application terminates), the implementation contains a clean-up step that releases the network interfaces that have been previously acquired. We do this by setting up a signal handler for the `SIGINT` signal, which frees up the acquired interfaces of the application before exiting the process.

The main functionality is setting up the ingress and egress interfaces, and transferring packets between them. While solutions to achieve this behaviour can be achieved using built-in Operating System utilities (e.g., Linux *bridge-utils* suite), implementation at the application-level is a fundamental building block in creating other microservice VNFs, as it does not rely on any underlying hardware and software assumptions. Furthermore, it provides a baseline for performance evaluation of increasingly complex VNFs.

The only runtime configuration options required are the two network interfaces that are designated as ingress and egress. Because various systems do not have a uniform naming scheme for their network interfaces, and to enable operation under heterogeneous environments, the decision to use runtime configuration of network interfaces is essential.

4.6.4 Lightweight Firewall

Firewalls are the most popular type of security-focused VNFs employed in today's networks. In the proposed lightweight model, the "wire" VNF described in Section 4.6.3 is expanded to contain basic L3/L4 packet matching functionality.

Headers of L3 and L4 (IP and Transport Protocol) network packets are at known, fixed locations and can be used for fast match actions with minimal processing overhead. Conforming to the context-based security paradigm [157], the functionality is designed to allow the majority of traffic, with DROP functionality specified by the operator depending on the client's requirements.

[11] https://www.juniper.net/documentation/en_US/junos12.3/topics/usage-guidelines/interfaces-configuring-the-media-mtu.html Retrieved January 2022

[12] https://www.arista.com/en/um-eos/eos-support-for-l3-mtu-on-7280r37500r37800r3 Retrieved January 2022

Code Listing 5 Basic firewall implementation

```
1  //...
2  #ifndef FW_RULES
3      #warning "FW_RULES not defined."
4      #define FW_RULES ""
5  #endif
6  #define DO_FW(pkt, FW_RULE, ...) \
7      unf_fw(pkt, FW_RULE, ##__VA_ARGS__)
8  //...
9  do {
10     unf_recv(ingress_intf, packet_buf);
11     //Firewall NF. Inspect for header match
12     if ((err = DO_FW(packet_buf, FW_RULES)) != NULL)
13         fprintf(stderr, "[FW]: %s\n", err);
14     else
15         unf_send(packet_buf, egress_intf);
16
17 }
18 while (1);
19 //...
```

The implementation uses preprocessor definitions for static embedding of rules at compile time, minimising the need for runtime configuration. We show the essential elements of the implementation in Code Listing 5.

In the Listing, Lines 2-5 provide some compilation-time checks, generating a warning that the VNF is misconfigured, and provides a fallback option for that eventuality. Line 6 defines the C preprocessor macro that is required for firewall operation. This macro is used to build a compile-time linked list of the firewall rules. The preprocessor definition splits inputs into individual rules, using the `__VA_ARGS__` element. Each rule is then converted to a list of constituent element tags (e.g., IP, TCP, UDP header analysis, and address or port match rule), which is then mapped to a binary representation of the pattern match, in network byte order, at compile time by benefiting from aggressive compilation optimisations.

Comparatively, the runtime packet matching functionality is straight forward. For each firewall rule, the associated packet header is selected by calculating the offset of the header start and end from the base packet address, and compared using XOR operations with the binary representation of the firewall rule. The main execution loop of the network function, Lines 9-18 of the Listing, receive packets from the ingress interface and perform packet matching. If the packet matches one of the firewall rules, then Line 13 prints an alert with the contents of the matched rule and the packet is implicitly dropped. Otherwise, Line 15 is executed, with the packet being

transmitted on the egress interface.

Once again, a signal handler for `SIGINT` which releases acquired network interfaces before terminating the application is implemented. The complete implementation, including preprocessor macros, mapping, binary rule generation and matching is achieved in under 300 lines of code.

Because this implementation only inspects the first Layer-specific header of the packet, it has certain assumptions and limitations in operation. The VNF assumes that the packets it receives are correctly steered by the network: a Firewall which checks IP(v4) rules does not expect IPv6 traffic. Furthermore, the implementation currently lacks support for encapsulated packets (e.g., IP in IP, 6in4, VXLAN, etc.). Further development of the solution can implement the functionality required for increased network function robustness and support multiple networking standards.

4.6.5 Deep Packet Inspection

Another popular category of security VNFs is related to Deep Packet Inspection. These services often perform a thorough analysis of entire packet payloads, matching against known signatures to detect and prevent specific attacks. Popular instances of DPI are found in IDS/IPS.

Code Listing 6 Basic DPI Implementation

```
//...
#ifndef IDS_RULES
    #warning "IDS_RULES not defined."
    #define IDS_RULES ""
#endif
#define DO_IDS(pkt, IDS_RULE, ...) \
    unf_dpi(pkt, IDS_RULE, ##__VA_ARGS__)
//...
do {
    unf_recv(ingress_intf, packet_buf);
    //DPI NF. Inspect pkt contents
    if ((err = DO_IDS(packet_buf, IDS_RULES)) != NULL)
        fprintf(stderr, "[IDS]: %s\n", err);
    else
        unf_send(packet_buf, egress_intf);
}
while (1);
//...
```

Unlike firewalls, these VNFs regularly rely on pattern matching algorithms, as the signatures may be found anywhere within the packet payloads. The most common algorithm found in DPI Engines is the Aho-Corasick algorithm [3], which performs matching against variable patterns; it is also commonly encountered in other domains, such as regular expression matching.

The behaviour of the generic algorithm includes specific provisions for wildcard matching that can include multiple random elements between patterns. While wildcard matching is a flexible tool in compressing multiple rules, it is rarely used in packet inspection proposed by the context-based model and microservice architecture. As a consequence, the implementation of a lightweight DPI Engine omits this feature, leading to more efficient runtime behaviour.

Code Listing 6 gives a high-level overview of the implementation of the DPI Engine in the proposed fashion. The overall code structure is similar to the firewall implementation presented in Code Listing 5: Lines 2-5 performs input checking, Lines 6-7 present a pre-processor macro for passing a list of IDS rules to the inspection engine, and Lines 9-17 highlight the main processing loop of the network function.

The structure of the DPI rules is considerably more complex than that of Firewall rules. Pattern data can take many forms, from Protocol source and destination ports to bytes of arbitrary length. Furthermore, the Aho-Corasick data structure, a trie with backlinks, can become increasingly complex. For this reason, the DPI codebase departs from pure C implementation. The language extensions of the C++11 standard, especially the `constexpr` specifier, make the development effort of the DPI solution more readable and maintainable.

The implementation tokenises individual DPI rules to identify header information and detection payload. Each token is mapped as follows: the packet header information (e.g., TCP destination port) is encoded within a tree structure using a mapping from well-known header fields (e.g., protocol version, ports, etc), similar to the firewall solution. Leafs of the tree point towards an Aho-Corasick trie. The payload inspection element is then converted to an Aho-Corasick trie. If an existing header is present within the tree, then the trie is readjusted to include the new payload within it.

Detection is done by initially traversing the header tree based on the information encoded within the network packet. If a header is matched through tree traversal, the trie associated with possible payloads is used for further inspection. If a pattern is present within the packet, then a VNF alert is issued with the DPI rule, and the packet is dropped. The complete implementation, because it uses two steps, with associated data structures, consists of approximately 1000 lines of C++-compliant code.

Parameter	Description
P	Number of packets processed
L	Size of packet payload data
R_H	Number of header-based processing rules
R_P	Number of payload-based processing rules

Table 4.2: Complexity analysis notation

4.6.6 Algorithmic Complexity and Overhead

The example network functions, although using the same software architecture, implement fundamentally different algorithms. Because of this, the network function implementations presented in throughout provide differential overheads.

Instead, of a direct comparison between the functions, an analysis of the algorithmic complexity for the different implementations is provided. In this analysis framework, we use the notation presented in Table 4.2.

The *wire* network function is the simplest to analyse: it performs no packet processing aside from reception and transmission. Thus, its algorithmic complexity is $O(P)$, which scales linearly with the number of packets being processed.

The *firewall* function only performs filtering based on packet header information. Because comparison between the defined rule and packet header requires a fixed number of operations, the required number of operations is $O(P * R_H)$.

Estimating the complexity of the *DPI* network function is best performed by decomposing the two steps through which packet processing is performed: tree traversal and Aho-Corasick runtime. For the latter, because the rules are known in advance and the data structure is computed in advance, the runtime is linear with respect to the number of payload rules and input size $O(P * (R_P + L))$, under the consideration that no more than one payload occurs in a packet. For tree search, the number of operations, in the worst-case is $O(P * R_H)$. Assuming the worst-case scenario, where each rule within a ruleset has a unique header description, but the same payload match the number of operations is $O(P * (R_H + R_L + L))$.

This analysis confirms that the performance of a VNF is impacted most by the number of packets that it has to process, as identified by several authors [38], [157], [97], [81]. Secondly, the complexity provides a rough guideline for network function performance, which agrees with existing literature [147], [58]: a overhead of a *wire* is less than the overhead of a *firewall*, which

in turn is less than the overhead of a *DPI*.

4.7 Summary

In this chapter, we have presented the implementation details of the security-oriented network function framework while maintaining the objectives presented in Chapter 3 for an Edge network architecture. In the network infrastructure, we realise the abundance of resource-constrained hardware and designed our framework to conform to heterogeneous environments.

We presented an SDN module that can be used to detect new application flows and request services following operator best practices and policies. The orchestration of network functions as presented in this context relies on several modules to maximise flexibility. The implementation aligns with the ETSI NFV references and looks at specific aspects pertaining to the detection, deployment, management and orchestration of security service chains required for modern networks.

In the next chapter, we propose a methodology for analysing the performance of individual aspects of the NFV paradigm and evaluate the constituent components of our framework under real-world scenarios.

Chapter 5

Evaluation

5.1 Overview

The evaluation presented in this chapter shows the main benefits of the proposed NFV framework and lightweight microservice design. In evaluating this work measurements taken from commodity NFV servers, resource-constrained hardware, and simulation results obtained using realistic edge network scenarios were used.

Section 5.2 shows the performance of microservice-based security VNFs, including experimental results on deployment times, memory consumption, composition of new services and changes in behaviour. Some experiments compare the microservice design with well-established monolithic architectures to justify the decision to use lightweight technologies introduced in Section 3.4.

Section 5.3 presents the results obtained from using the NFV management and orchestration framework described in Section 4.4. The performance of per-user creation, deployment, and modification of VNFs is compared to operator-defined static classes. Control network measurements including reaction to network events and delays incurred by performing on-the-fly composition of network service chains are compared with production-grade NFV frameworks, such as Kubernetes.

Finally, in Section 5.4, the benefits of the proposed Latency-Optimal Placement of Service Chains optimisation model presented in Section 3.6 and associated heuristic Minimal Path Deviation Algorithm from Section 4.5 are analysed. First, the latency benefits of on-path placement

of network services are compared to localisation at centralised locations (e.g., data centres). The performance of the heuristic algorithm is investigated with respect to number of latency violations encountered by users, global network latency, and link utilisation rates. The evaluation is concluded by an analysis on the robustness of the proposed algorithm in harsh network environments, where individual link latencies are higher than those encountered in edge networks.

5.2 Performance of the Lightweight Network Function Architecture

This section highlights the most important characteristics of lightweight NF modules that were measured on a commodity NFV server and resource-constrained devices.

The evaluation environment used is described in Section 5.2.1. NF activation times, a crucial performance element when considering dynamic network scenarios, is presented in Section 5.2.2. As the intended operation is within edge networks, where VNF Hosts primarily consist of resource-constrained devices, the packet processing overheads are evaluated in Section 5.2.3, and memory residency is investigated in Section 5.2.4. Achievable throughput of the VNF architectures is analysed in Section 5.2.5.

The contribution of these measurements is to compare microservice architectures with traditional monolithic designs and to present the properties required for next-generation VNFs, such as delay introduced, chaining properties, and activation times. Due to their tailored nature, bespoke designs which conform to a given network architecture prove difficult to evaluate in a heterogeneous environment, as presented in this section.

5.2.1 Evaluation Environment

The following measurements were conducted using a mid-range computer with a commodity Intel i7 CPU and 8GB DDR3 RAM. The operating system used was Ubuntu 18.04 LTS with Linux kernel version 4.15.

Measurements involving edge devices were performed using a commodity resource-constrained device, a Raspberry Pi 3B, with an ARMv8 CPU and 1GB RAM. Raspbian OS (a Debian-based OS) was installed, with Linux kernel version 4.14.

In terms of NFs evaluated, DPI Engines were used, as they are among the most complex network services employed in real-world networks in terms of activation times, processing overhead, memory residency, and achievable throughput. The well-known Snort DPI engine, version 2.9.8.3, was chosen as it conforms to the monolithic software architecture, with single-threaded packet processing capabilities.

In order to accurately measure the performance metrics, the applications were hosted natively on the hardware, without employment of any virtualisation or containerisation technologies. The cumulative overheads from Virtual Machine or Container allocation, initialisation, and startup would be equivalent for the measurements conducted, and would thus cancel out.

The rulesets used for evaluation were identical between the different software architectures. The rules were randomly selected from the Snort Registered ruleset snapshot *2983* which contains approximately 50,000 rules designed by Talos Security (a subsidiary of Cisco) to protect against real-world threats.

5.2.2 Start and Reconfiguration of Microservices

In order to provide high flexibility in deployment and migration of VNFs in networks, the time between when the NF application starts initialising to becoming operational, able to receive and process network traffic, is a crucial factor.

We measure the time from the application starting to it starting to receive packets using the *ts*[1] utility with microsecond precision. This time includes the loading of the application, run-time configuration (e.g., Snort builds the associated data structures for detection during this time), network interface acquisition, and finally receiving packets into an internal buffer. Figure 5.1 shows the time required for microservice-based NF architectures to become active versus traditional monolithic software designs.

In order to change the traffic patterns that are being inspected, the Network Function application requires a process that reconfigures the behaviour, by stopping packet processing, freeing the underlying data structures, re-reading the inspection rules, and populating new data structures that contain the new information. Reconfiguration of a microservice requires stopping the existing instance and replacing it with a new one that contains the desired behaviour. On the other hand, monolithic architectures rely on run-time reconfiguration through configuration files that need to be re-parsed; for example, Snort can be reloaded using the steps described above by

[1] http://joeyh.name/code/moreutils/ Retrieved December 2021

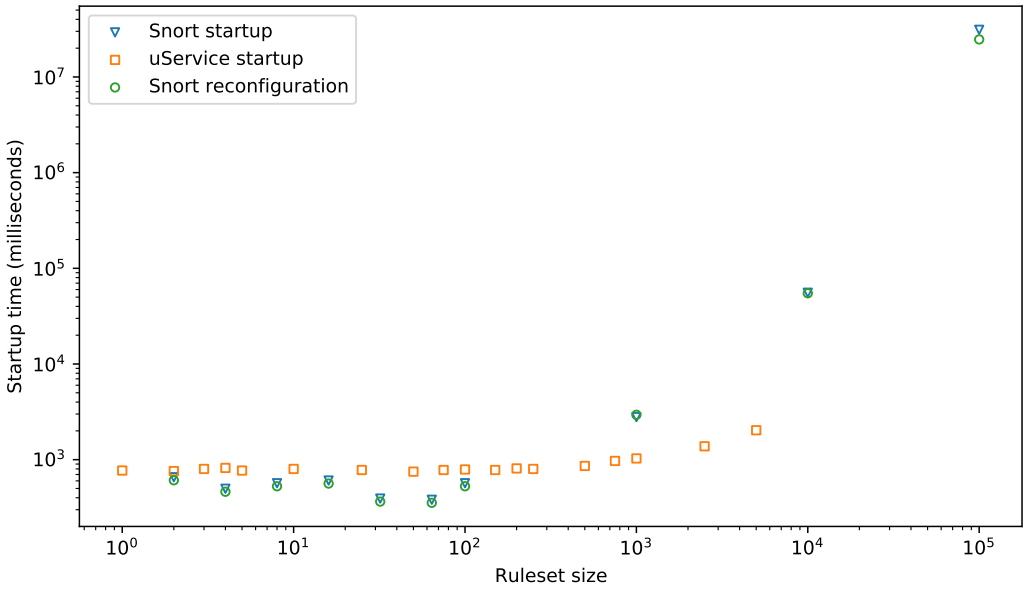

Figure 5.1: Activation and reconfiguration time required for VNF architectures

implementing a signal handler for the SIGHUP message.

Use of lightweight NFs is often preferable due to their rapid and predictable times to activation. In the scenarios presented, however, the microservice-based VNFs are already available as software artifacts; investigation into the consequences of on-demand compilation of VNFs is performed in Section 5.3.2.

5.2.3 Packet Processing Overheads

Keeping processing overheads low is important for transparent network services. It is a key benchmark when considering new VNF designs. Comparing different architectures is challenging, especially in heterogeneous hardware environments. However, as argued in Section 3.4, overheads incurred by the complexity of different architectures can be observed by investigating the scaling behaviour.

In the experimental setup, both monolithic and microservice versions of the system are executing the same functionality (e.g., detection of pre-defined packet payloads). The underlying DPI engines are based on the Aho-Corasick algorithm for pattern matching [3], and packet acquisition is performed in a similar manner. For the purposes of this evaluation, traffic generated using iperf is separated into benign, without containing the signatures used for inspection —

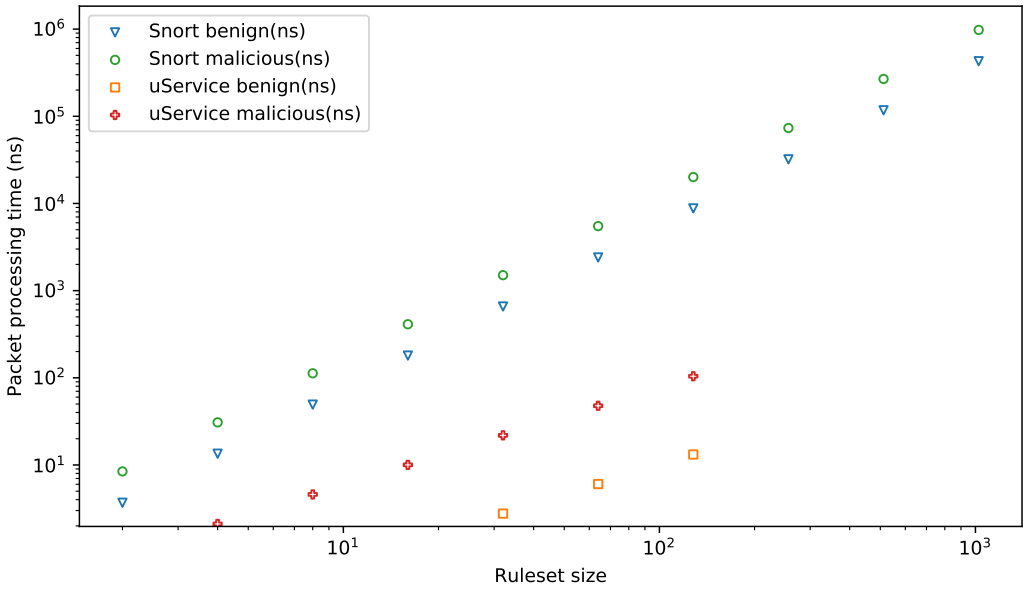

Figure 5.2: Packet processing overheads

thus no packet processing should occur —, and malicious, to analyse the performance impact of the services when packet processing is needed.

Figure 5.2 compares the delay introduced by different architectures through measurement of packet ingress/egress times when traversing a VNF. Snort delays for benign traffic (that does not require explicit inspection) are unnecessary. The added overhead is due to the fact that, for all classes of network traffic, the monolithic architecture has to identify at run-time whether a packet is subject to inspection at packet ingress. Microservices target only a subsection of traffic; even on 'suspected' traffic, the packet processing overhead is lower than the monolithic variation because of the architectural exclusion of dynamic classification. The packet classification component is integrated within the network data plane (e.g., through SDN flow tables), being redundant in monolithic architectures.

5.2.4 Memory Requirements

NFV servers in network edge scenarios have limited memory on board. For example, commodity routers have as little as 512MB of available memory. Minimising the memory footprint of a VNF is important, as it can allow resource-constrained devices to have increased VNF capacity. In Figure 5.3, the runtime memory usage of the NF architectures is compared.

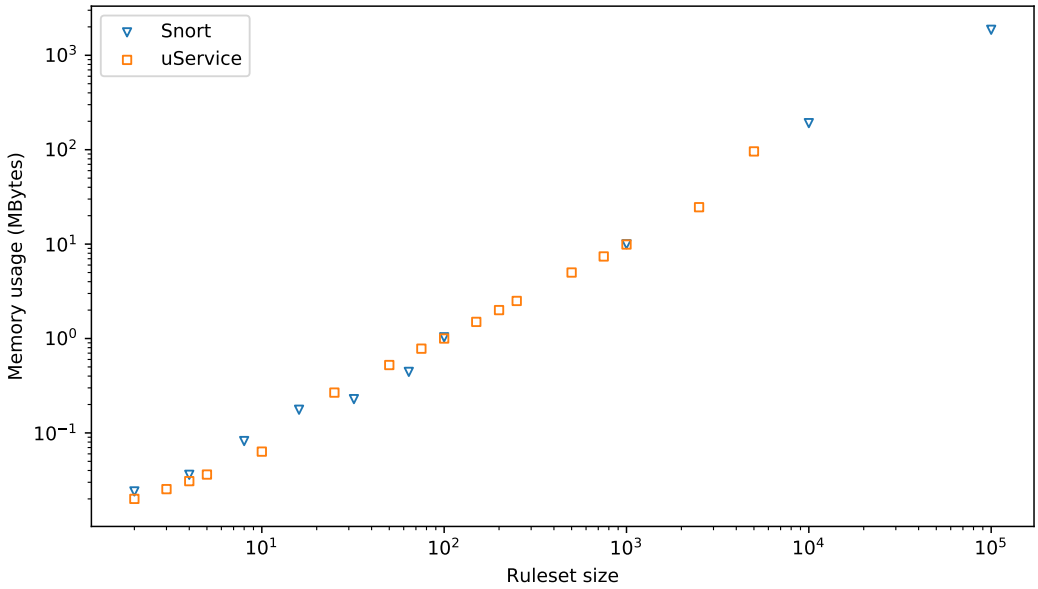

Figure 5.3: VNF memory consumption

Overall, memory consumption of the two systems is comparable. The main variation is dictated by the Aho-Corasick algorithm data structure [3]. Remaining architectural elements (e.g., packet decoding and preprocessing) become less relevant with respect to memory requirements when larger rulesets are analysed.

Monolithic designs use run-time configuration files that, when parsed, dictate the memory consumption. There is no prior knowledge to the system resources that need to be dedicated to hosting a VNF instance, as the static memory dedicated to the VNF is not indicative of the runtime needs. Through techniques such as static analysis, the memory requirements of microservices can be more easily estimated.

5.2.5 Achievable Throughput

In an environment similar to the one used in Section 5.2.3, iperf was used to measure maximum throughput between source and destination connected through running DPI VNFs and 'wire' VNFs (simple bridges connecting ingress to egress interfaces, as shown in Section 4.6.3). Throughput has been measured in Mbps, as the functionality of the VNFs evaluated is tightly coupled with the data rate transmitted, and not the number of packets that are being processed.

Results in Figure 5.4 show maximum achievable throughput on different systems acting as

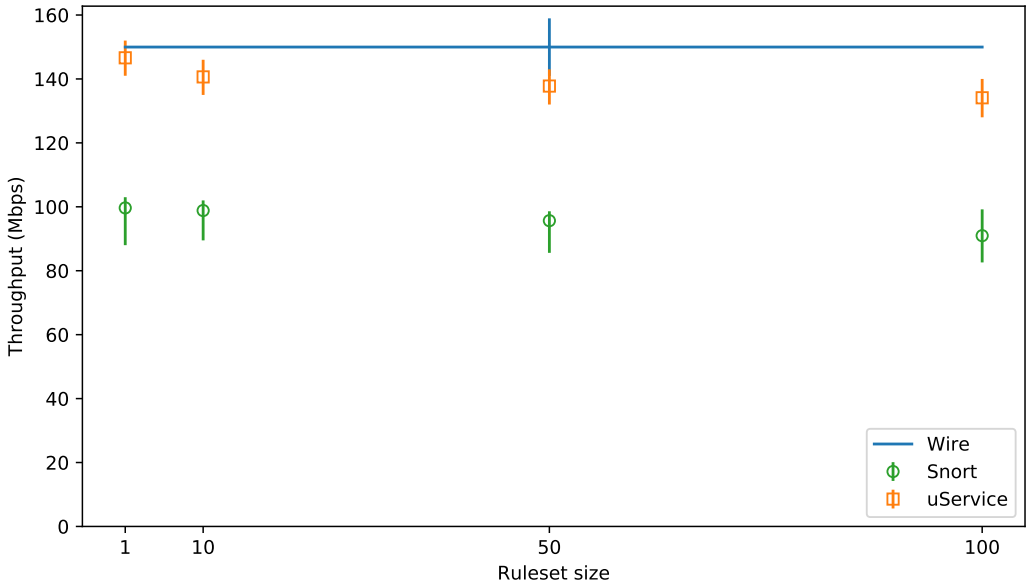

Figure 5.4: Throughput of VNFs on different hardware

VNF Hosts, and is limited by the speed of the network topology. In the case of the resource-constrained Raspberry Pi, it only benefits from a singular 100 Mbps network interface that is being used for both packet ingress and egress.

Even under ideal circumstances, monolithic applications do not efficiently utilise the available network resources, thus creating artificial congestion throughout the network, introducing packet losses, and increasing the delay experienced by end-users. Combined with our observations in Section 5.2.3, we believe that the added complexity of unnecessary dynamic components (which, as previously mentioned, have been offloaded to the network data plane), is the main factor for the loss in throughput which is observed in Figure 5.4. By limiting the scope of microservices to only perform inspection on targeted traffic, they can make use of the available resources to improve throughput and capability of processing packets.

5.3 Responsiveness of Per-Device Service Chains

We designed and implemented our security NFV solution to be able to operate at the network edge, which is a highly dynamic environment that necessitates rapid responses in function deployment due to client behaviour. As a result, we propose the evaluation of our framework when changes within the network occur (e.g., clients joining and leaving), to better emphasise the need

for these properties.

5.3.1 Evaluation Environment

The network environment for the evaluation of the NFV framework is achieved through emulation using Mininet. In obtaining the results described in this section, the well-understood Docker containers which are used in other proposed lightweight NFV frameworks [34] [134] [69], are used to host the VNFs. Orchestration of these containers is performed using the Kubernetes framework, which allows for implementation of the custom behaviour described in Section 4.4.

The evaluation for this section was performed using commodity servers, equipped with an Intel i7 CPU and 16GB DDR3 RAM. The operating system used was Ubuntu 18.04 LTS with Linux kernel version 4.15. This hardware setup is typical of a server hosting NFV Management and Orchestration components within real-world edge network deployments.

5.3.2 Creation of Targeted Microservices

The ETSI NFV architecture [42] recommends using a VNF store to transfer the services onto capable hosts. However, using such a mechanism with the pre-built microservices proposed in Section 4.6, the storage space required becomes unrealistic.

In Section 4.6 we presented an On-The-Fly (OTF) build system that creates targeted microservices with the desired security behaviour. We thus allow operators and clients alike to use a common framework to express security policies. We investigate the time required for the NFV framework to create the desired software artifact through the build system and transfer the required VNF to its intended server. We compare this to the use of a dedicated VNF store, which contains pre-built software artifacts in Figure 5.5.

Compared to a dedicated VNF store, the performance at first glance is impacted. To this end, we use one target architecture (x86) and increase the number of rules used for creating the relevant VNFs. As can be seen in Figure 5.6, the storage requirements for using a VNF store increase exponentially with the number of individual rules, quickly going into the Exabyte range. This behaviour is expected, because of the number of ruleset combinations that need to be achieved; as a simple example, if we have a rule set $\{A,B,C\}$ then the resulting VNF artifacts would require the following rules embedded $\{A,B,C,AB,AC,BC,ABC\}$, each within an individual ar-

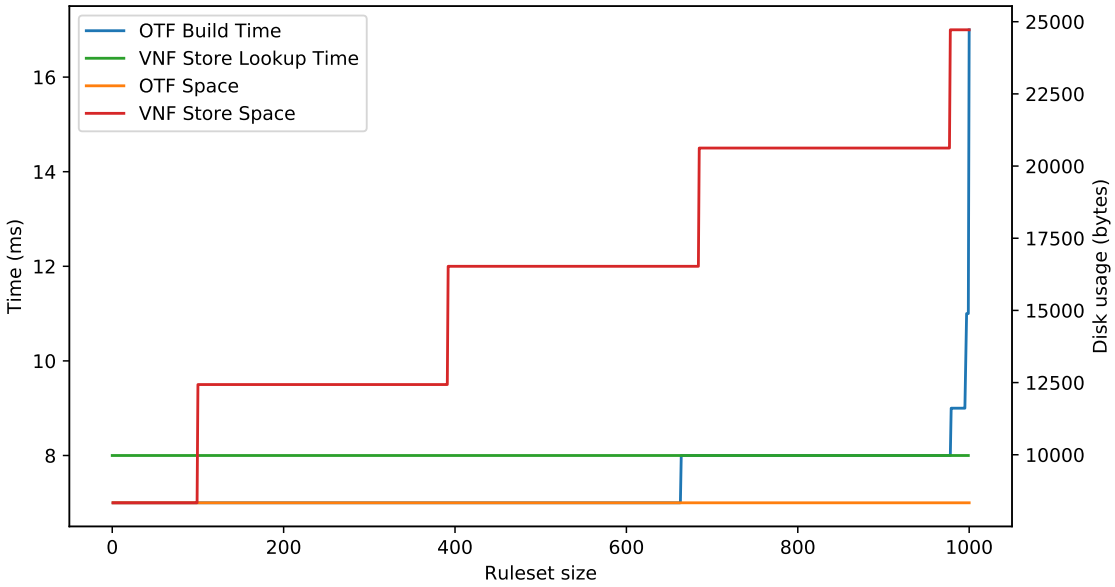

Figure 5.5: Build times and artifact sizes

tifact. Thus, the number of VNFs (and the resulting artifact storage space) increases according to the mathematical binomial coefficient formula $\sum_{k=1,n} \binom{n}{k} = n! \sum_{k=1,n} \frac{1}{k!(n!-k!)}$ which can be further simplified to $\sum_{k=1,n} \binom{n}{k} = 2^n - 1$, where n is the total number of packet processing rules the operator wishes to employ. In our scenario, we do not factor in the empty set as it would result in a simple Wire VNF without any functionality. Considering the characteristics of edge network environments the solution is targeting, this increase can limit the practical applicability of the proposed framework.

5.3.3 On-demand Deployment of Service Chains

Service Chains in networks traditionally service multiple clients, with traffic steering rules defined in advance. Context-based security function management is most effective when each device is served by its own dedicated VNFs because the logical separation between devices allows for rapid function updates in response to emerging malicious activity.

Providing transparent network services requires minimal spin-up time, to incentivize adoption of context-based security without affecting end-users. We look at the time required for our NFV framework to detect and identify the context of a device, to deployment of service chains of varying length determined by the context. We compare the overall overhead with a well-known NFV service orchestrator, Kubernetes, which is widely deployed in existing networks.

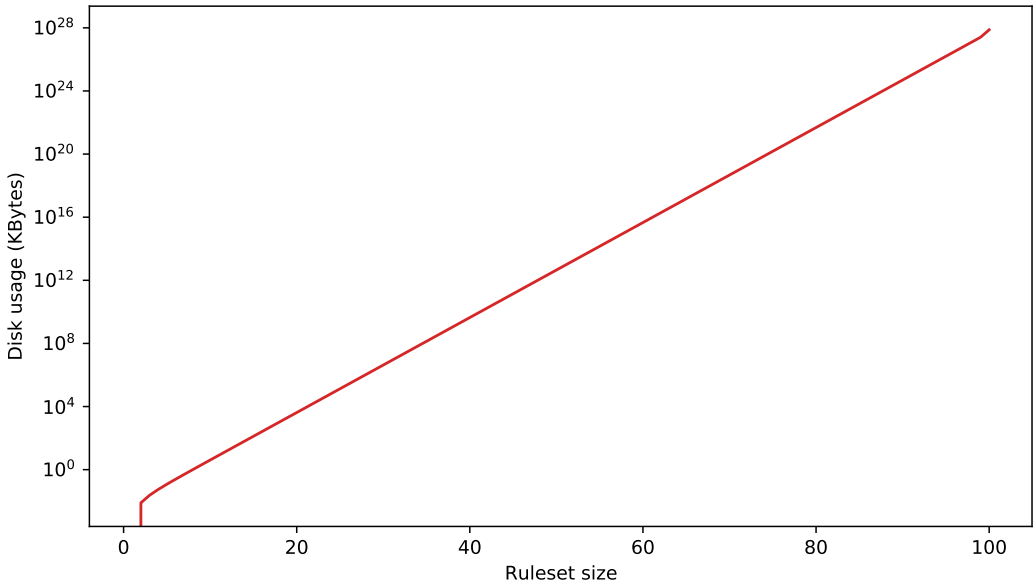

Figure 5.6: Storage space requirements for μservice VNF variations

We illustrate our findings for VNF activation in Figure 5.7.

A detailed breakdown of the delays incurred by each component are presented in Figure 5.8. We compare in this scenario the delays in completing deployments of service chains of different lengths using both our proposed solution and equivalent behaviour found within Kubernetes. The overall overhead added by the proposed detection and context identification components is kept to a minimum, with the majority of delays being attributed to the communication overhead between multiple systems in the framework. By employing the OTF build system, we also reduce the transfer overhead, thus providing improved responsiveness compared to state-of-the-art solutions. This behaviour is observed due to VNF hosts under Kubernetes transfer an entire container (e.g., a multi-layered Docker image) and initialises it, as with our solution, the OTF build system produces a single artifact that is much smaller in size.

5.3.4 Autonomous Lifecycle Management

The highly dynamic nature of edge networks means that control networks risk becoming congested with messages pertaining to lifecycle management. Knowing that individual functions in a security service chain might drop network traffic and render other functions as idle, we proposed in Section 4.4 the autonomous lifecycle management scheme in order to increase resource availability and reduce risk of congestion of the control network.

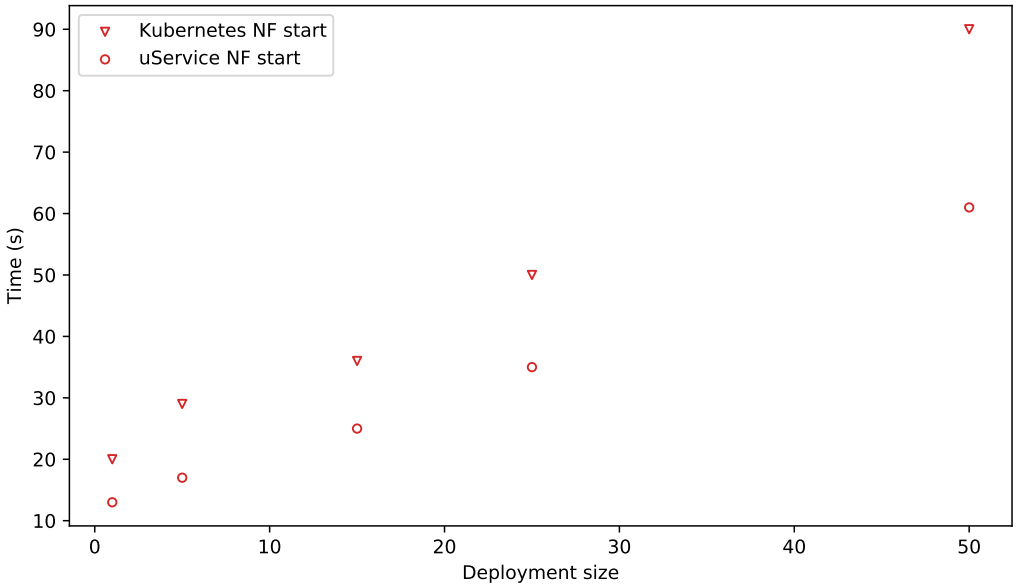

Figure 5.7: Deployment using different framework and service architectures

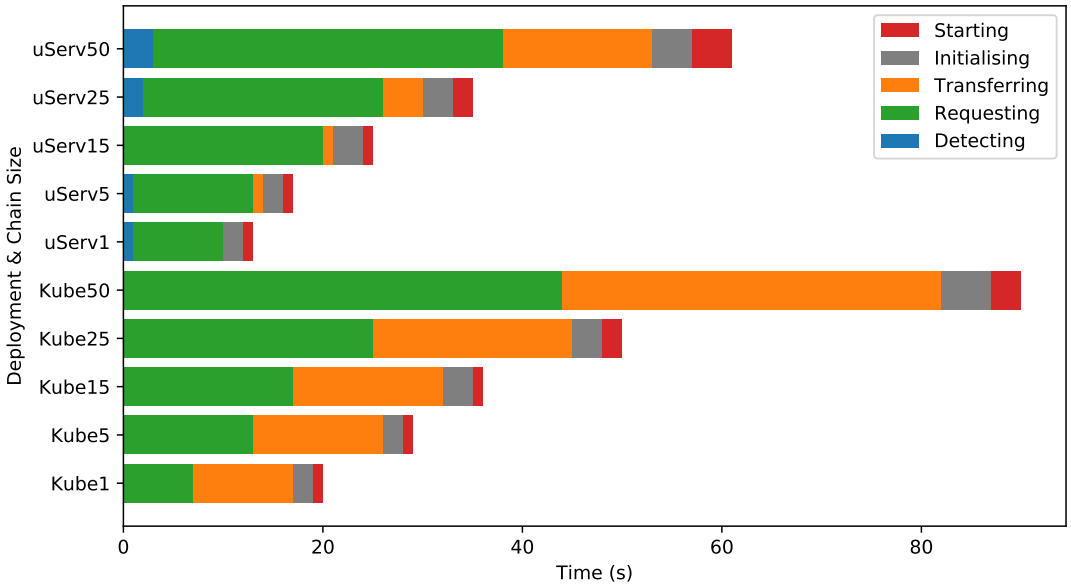

Figure 5.8: Analysis of individual component delays in deployment

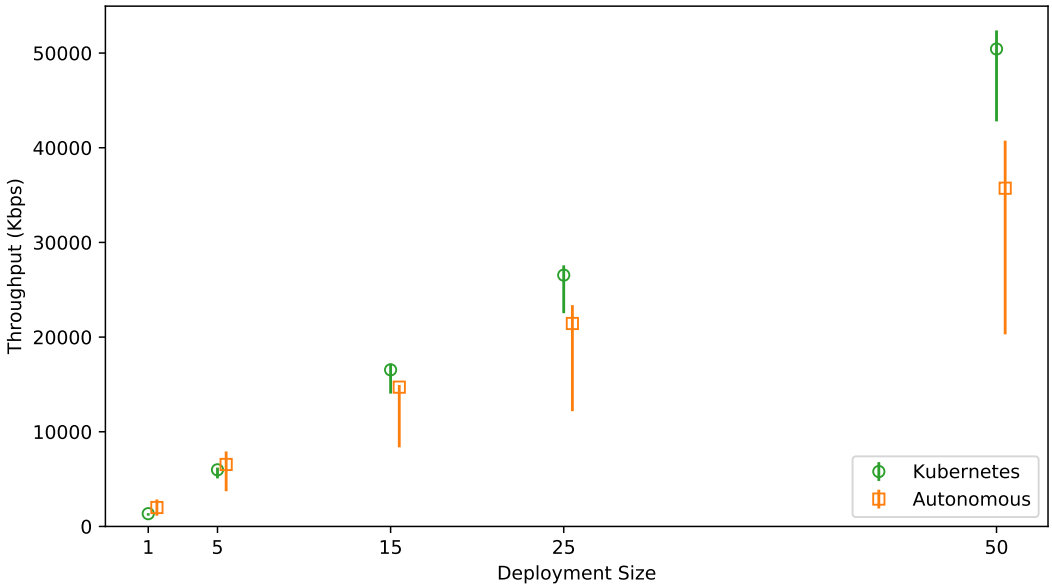

Figure 5.9: Investigation on control network resource usage

We compare the autonomous lifecycle management scheme, which lets individual VNF hosts determine the lifecycle of the Network Functions they are hosting, to the one used by Kubernetes, a production-grade orchestration system. In the latter scenario, the Management and Orchestration controller is responsible for issuing lifecycle commands, increasing the NFV control plane overhead. We look at the number of messages exchanged by the NFV controller and VNF Host when performing multiple VNF allocation and de-allocation requests, and present the findings in Figure 5.9. Due to the autonomous nature of the lifecycle management system, we reduce the overall control plane communication while maintaining essential health monitoring of the VNF Host infrastructure.

5.4 Latency-Optimal Placement of Service Chains

For placement of service chains using the context-based security paradigm, which requires non-sharing of VNFs between clients, we have implemented the Minimal Path Deviation (MPD) allocation algorithm as described in Section 4.5. To show the properties of such a system, we have designed a series of experiments based on a simulated environment[2]. After presenting our proposed experimental environment (Section 5.4.1) that was modelled using a real-world topology and network-level measurements, we evaluate placement of our Context-Based Optimal

[2]https://github.com/mirceaIordache/ChainingILP

Solution (as presented in Section 3.6) and a state-of-the-art VNF embedding strategy as presented by Dietrich et al. [39]. We show the emerging latency benefits of using in-path placement of service chains in Section 5.4.3. We further analyse the properties of the proposed algorithm with respect to traffic steering in Section 5.4.4. We finish our evaluation by introducing temporal latency variation and present the robustness of the placement scheme in Section 5.4.5.

5.4.1 Experimental Setup

5.4.1.1 Network Topology

As a basis for the network topology used, we have used the nation-wide Jisc NREN backbone network, as reported on Topology Zoo[3], and it contains 28 nodes and 45 edges. A visual representation of this topology is also presented in Figure 5.10.

In order to approximate edge resources, we have assumed finite resource availability at all points of presence in this network topology, with each server able to host a limited number of VNFs. The deployment where computational availability is present alongside each network device is in close alignment with the ETSI MEC suggested scenario [64]. We have introduced two Cloud Data Centres to be used, to represent the internal NFV infrastructure with unlimited capacity for hosting of VNFs. The two Data Centres were selected to be centrally located within the topology with the minimal average number of hops to any other given node. Users have a direct connection to one of the nodes within the topology, selected at random.

5.4.1.2 Application Modelling

We have categorised the applications based on the expected maximum tolerance level of end-to-end latency of packets, giving our summary in Table 5.1. We assign a total latency $\theta_{s,d}$ for each request $\mathbb{N}_{s,d}$ representing the maximum allowed latency for the flow, beyond which the user notices application performance degradation. This information is derived from work showcasing the benefits of next-generation networks and the envisioned applications [62] [28] [128].

In terms of bandwidth consumption, we use three classes of applications, which are described in Table 5.2. The resulting model thus consists of applications selected from a feature matrix for the two principal network characteristics.

[3]http://www.topology-zoo.org

Figure 5.10: Topology Zoo JANET Backbone Topology

Application type	Latency SLA
Hard Real-time (e.g., autonomous vehicles)	5ms
Soft Real-time (e.g., AR/VR)	10ms
Near real-time (e.g., video conferencing)	30ms
Non real-time (e.g., data transfer)	100ms

Table 5.1: Application classes and their latency requirements

Application type	Description	Bandwidth Demand
Control-only	Online gaming, telemetry streaming, etc.	50Mbps
Compressed Video	Conference calls, social media	100Mbps
High-Quality Video	Streaming services, Full remote control	200Mbps

Table 5.2: Application classes based on bandwidth consumption

Network service requests are generated based on real-world network security applications (e.g., Access Control List, Firewall, Intrusion Detection or Prevention) [95]. The individual NF computational requirements are derived from resource profiles for each class of NF (i.e., required CPU cycles per NF or memory footprint). Resulting request chains are thus comprised of a uniform distribution of NF selections from a set of three classes, each with its own resource requirements.

5.4.1.3 Link Latency and Bandwidth Modelling

End-to-end latency has been modelled using millions of end-to-end latency measurements from real-world applications. Data has been collected from the New Zealand research and education wide-area network provider REANNZ using Ruru [37]. Modelling has been performed in a similar manner to the process described in [32], with individual link latency values sampled from a Gamma distribution ($k = 2.2, \theta = 0.22$) to create a representative time series.

For link bandwidth, we allocate a bandwidth of 500Mbps for access links which connect users to the network topology, and 1Gbps for core links that provide connectivity between the topology points of presence (PoPs). The choice for the former is to ensure that congestion does not occur at the access level, which would skew results due to insufficient bandwidth during service requests. The latter choice of bandwidth arises due to the prevalence of commodity network devices (e.g., home routers, small business core switches, etc.) which provides equivalent characteristics. Finally, connections between nodes that are designated as Data Centres (DC) have a link bandwidth of 10Gbps to align with the high-capacity nature of the PoP. Higher capacity of the links in the proposed network topology is unnecessary; even at maximum utilisation of the 1Gbps links for inbound traffic, the DC links would not be fully utilised.

5.4.2 Optimal Placement Strategies

We first compare the viability of the Context-Based placement strategy advocated in this thesis against state-of-the-art VNF embedding solutions which focus on co-location of services. To this end we chose the Nestor Network Service Embedder presented by Dietrich et al. [39], in the Mixed-Integer Programming formulation, which provides an optimal allocation focusing on minimising the embedding footprint of services. Furthermore, we look at the impact of bandwidth constraints within the Context-Based placement formulation within edge environments.

The formulations for the two optimisation problems have been implemented using the Gurobi solver [57]. For this experiment, we consider an offline scenario, where network service requests are batch-processed, and we vary the batch size. By using this methodology we obtain unity acceptance ratio of requests but can only operate on small batch sizes, of up to tens of requests simultaneously.

The solver calculates the optimal allocations of service chains within the topology, according to the respective objective functions and constraints of each formulation. Based on these allocations, we calculate the total network latency for all flows which are serviced by resulting deployments. Our evaluation in this experiment focuses on the total path latency and the total path length that the flows encounter.

Furthermore, we investigate, under edge network conditions, the effect of utilising bandwidth constraints within the ILP formulation to ensure that individual links do not become oversubscribed. To this end we use the entire formulation of the ILP described in Section 3.6 as the Bandwidth-Constrained ILP, and a secondary formulation omitting Constraint (3.8), which avoids link oversubscription, as an alternative.

We perform a comparison between the Context-Based Placement strategies and the state-of-the-art Nestor NSE formulation by performing allocation of network services on residual resources distributed within the network, and minimising allocation footprint, respectively. Figure 5.11 represents the cumulative path latency and length with respect to the number of service chain requests being processed. Both formulations of the Context-Based Placement strategies consistently yield lower overall latencies and path lengths when compared to the more common VNF consolidation approach.

For the number of requests studied, the two Context-Based variations provide identical placement of services. This is related to the resource scarcity within the network edge, which also limiting the number of requests that can be processed within a given batch. Since the scope of

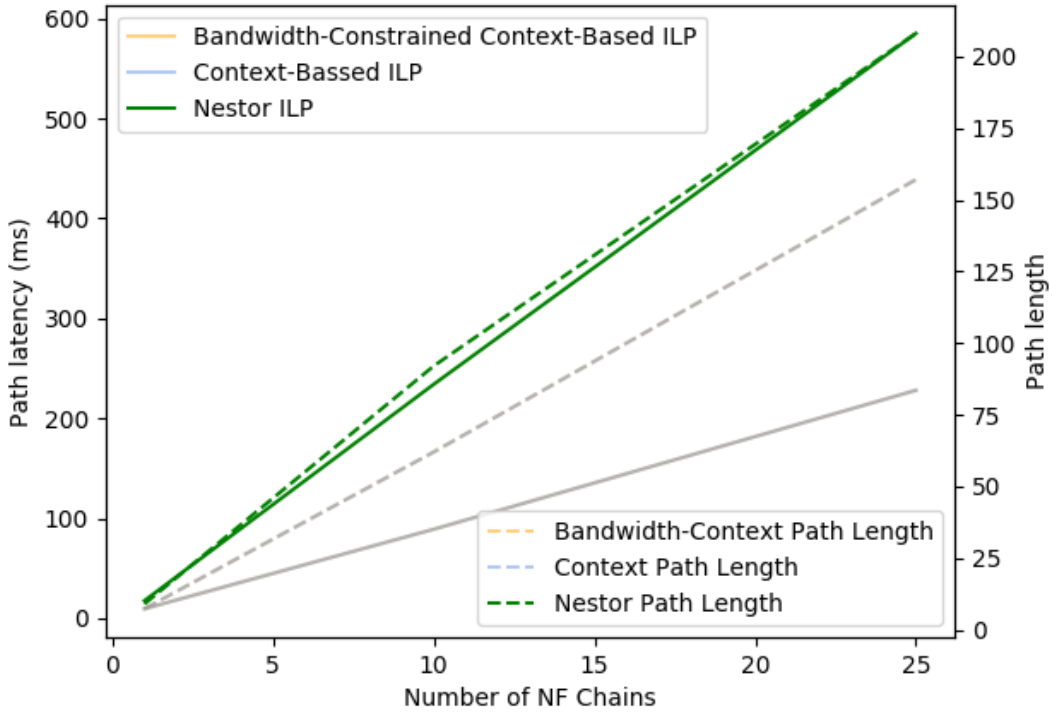

Figure 5.11: Cumulative path latency and length for optimal placement formulations

the work focuses on the resource-constrained nature of the network edge environment, an increase of the resource availability of edge PoPs (i.e., transitioning to a Data Centre model with uniform network-wide computational availability) is unrealistic.

In light of the observed results the choice of Context-based placement, whose goal is minimisation of end-to-end network latency and utilisation of residual resources within the network, within edge networks is preferable to alternative state-of-the-art solutions which focus on VNF consolidation. Furthermore, the impact of limited bandwidth in such networks is minimal, overshadowed by the resource-constrained nature of the environment.

5.4.3 Heuristic Allocation

Following the investigation into the placement results of the two paradigms, we compare the performance of the Heuristic Minimal Path Deviation (MPD). Figure 5.12 shows the cumulative latencies and path lengths for the MPD algorithm when compared to the ILP solution under the same circumstances as described in Section 5.4.2. As expected, the cumulative latencies of the heuristic are higher than an optimal solution, however, on average, we observe a performance

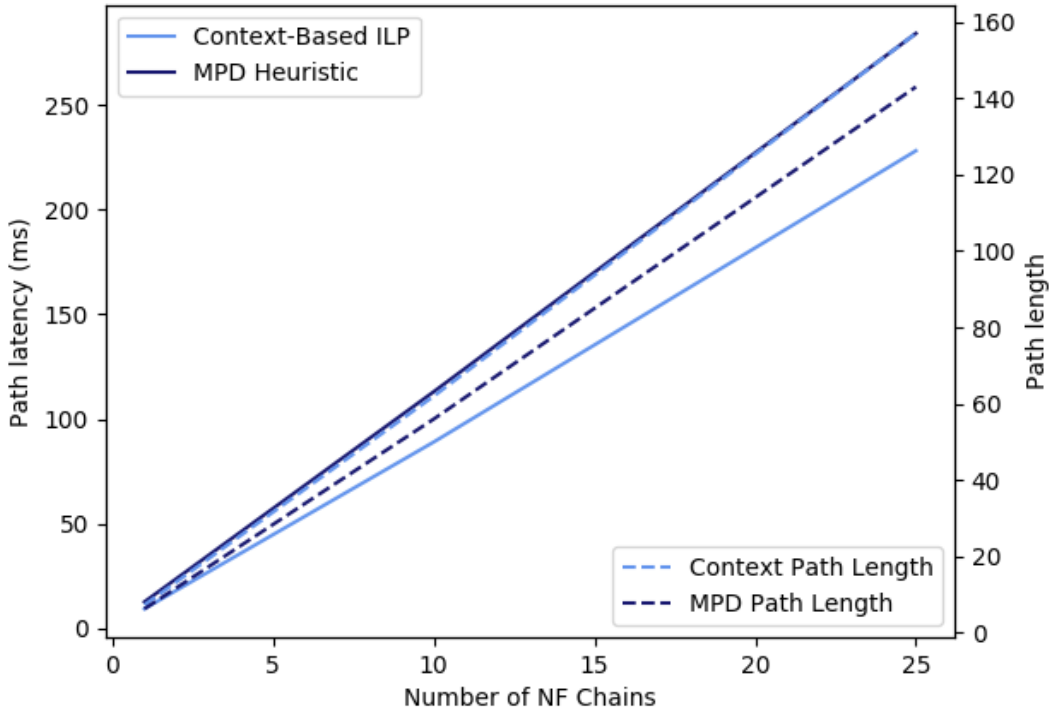

Figure 5.12: Cumulative path latencies and length in heuristic scenarios

increase of up to 50% when comparing against Nestor due to use of residual network resources.

5.4.4 Deviation from shortest and optimal path

Our MPD algorithm calculates the shortest path which contains the necessary hardware resources to fulfil a service request. In Figure 5.13, by considering a fixed number of Service Requests and varying the end-user locations within the network, we perform a comparison between the path lengths of the optimal and heuristic allocation models. We normalise these results with respect to the shortest path between the user and destination. Using longer paths for traffic routing in the optimal solution gives improved initial latency. However, in large networks, we argue that hardware limitations of switches become apparent, increasing the switching delay and leading to overpopulation of forwarding tables.

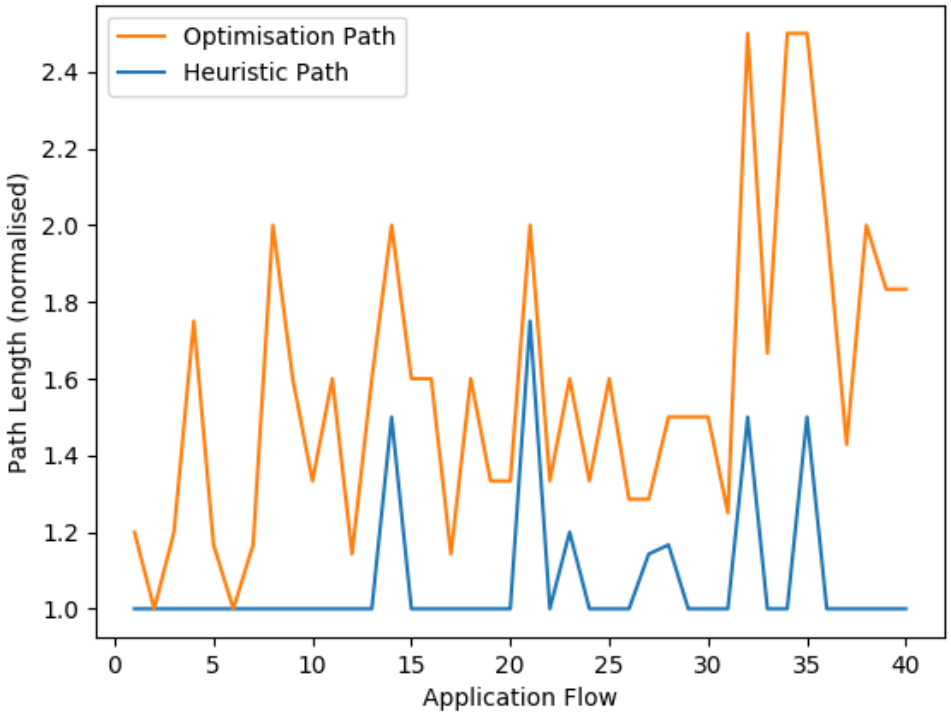

Figure 5.13: Path lengths compared to shortest network path

5.4.5 Dynamic Network Behaviour

As link latencies change over time (due to user utilisation), the placement of VNF is subject to temporal variations. These deviations from a previously optimal allocation may result in latency violations which note application performance degradation. We analyse the resulting violations to provide key insights into dynamic rescheduling of service chains.

This experiment expands on the one presented in Section 5.4.3 by introducing a temporal component that updates the latency matrix l every time instance t (such a time instance can be e.g., 1 minute in a real-world network). We then analyse the number of latency violations that occur over the given time period.

We perform our evaluation by considering a fixed number of requests, and compare the efficiency of initial allocations using the proposed algorithm with the optimal solution when hosts are placed randomly within the network. To do so, we analyse the number of SLA violations observed after one timestep instance (i.e., by introducing jitter within the network links). As can be seen in Figure 5.14, in the majority of cases, our proposed MPD algorithm performs optimal or near-optimal placement of service requests. The number of application requests that initially

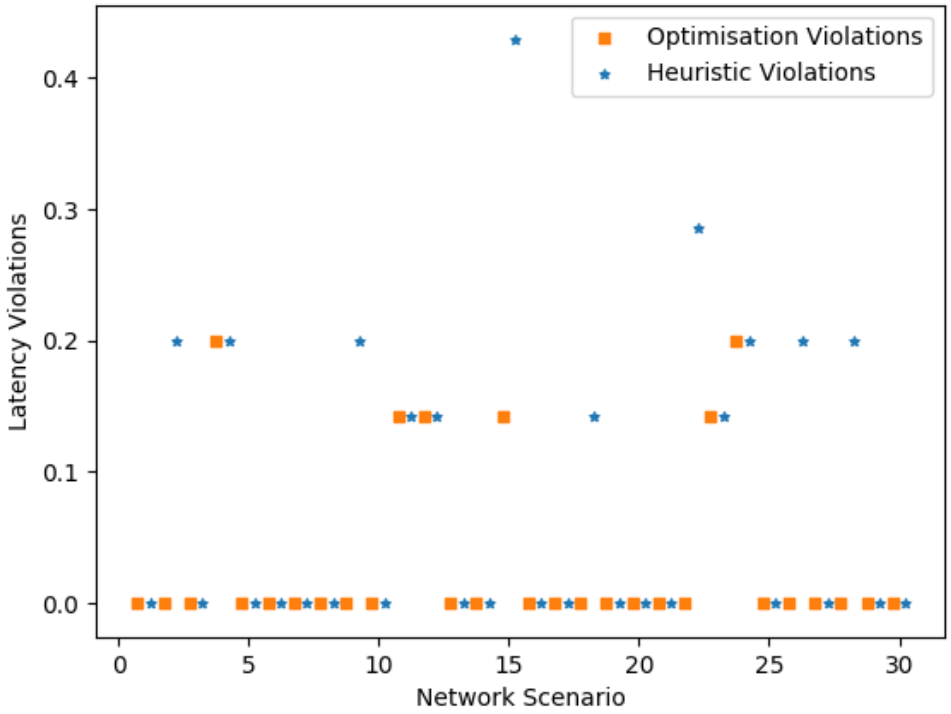

Figure 5.14: Initial placement efficiency

cause latency violations is 23.3%. Of particular interest is the scenario where an allocation causes 42.85% of the application flows to suffer latency violations. We have identified that the later service requests were allocated on longer paths because of the First-In-First-Served prioritisation model employed by the heuristic allocation process. In real-world situations, these types of issues can be mitigated by sorting the incoming requests by application latency requirements prior to allocation.

We compare the number of SLA violations when the number of requests changes. We run the simulation over 100 time instances and present the behaviour of our placement system. To this end, we present, in Figure 5.15 the total number of latency violations observed after 100 time steps. Of interesting note is that Nestor consistently generates a large number of violations due to routing all network traffic to one of the Data Centres within the topology. Once again, we also observe that the bandwidth constraint of the Context-Based Placement Problem does not produce any noticeable differences.

We perform a more granular investigation of the latency violations, by looking at normalised the per-flow violations occurring at every timestep. As shown in Figure 5.16, the number of latency violations occurring for the optimal placement is significantly higher than the violations generated by the MPD heuristic scheme.

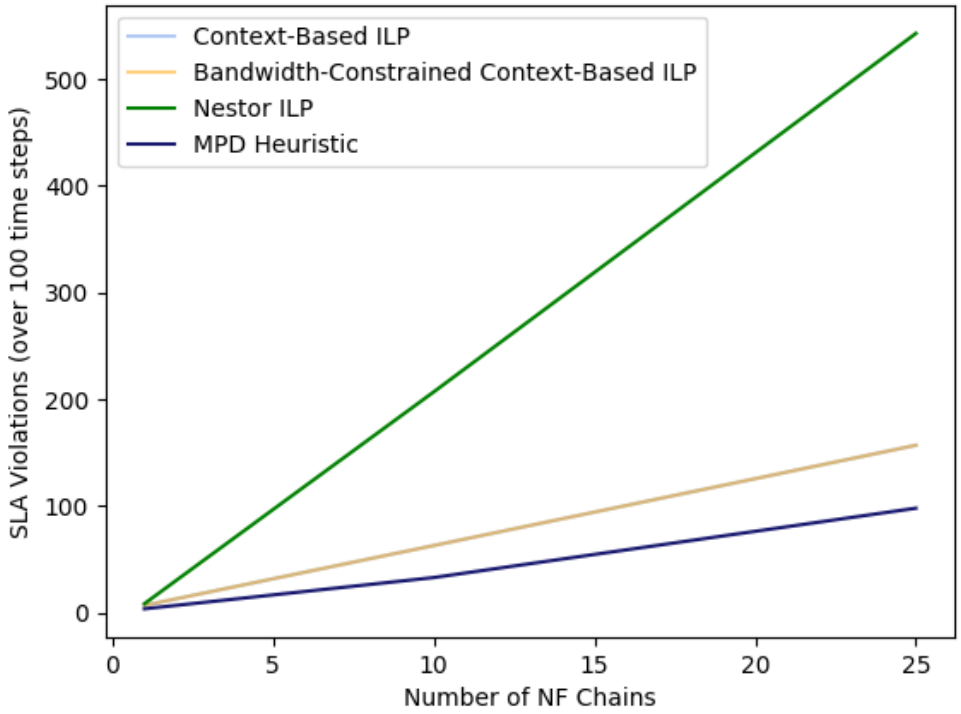

Figure 5.15: Latency violations after 100 time instances

We attribute this increased robustness to temporal latency variations to the shorter paths used in placement of services. On the other hand, the optimal allocation attempts to minimise the end-to-end latency at a given time instant, hence introducing multiple delay-sensitive elements within the chosen path. When also factoring in the time required for finding an optimal solution for the allocation problem, the heuristic solution is preferable, even if certain latency violations are still encountered.

5.4.6 Robustness of Heuristic

The MPD and Latency-Optimal VNF Chain Placement Optimisation Problem have been designed with network edge scenarios in mind. However, we consider the possibility of harsh network conditions where the network latency is similar to core and Internet scenarios.

In order to maintain the characteristics of the network traffic, according to [37], we maintained the same Gamma distribution characteristics, and scaled the resulting mean latency $k\theta$ by a constant α. For the latter, we used values from 2^0 to 2^{14} to analyse the behaviour of our proposed approach in adverse network conditions. Each link latency is independently sampled from a

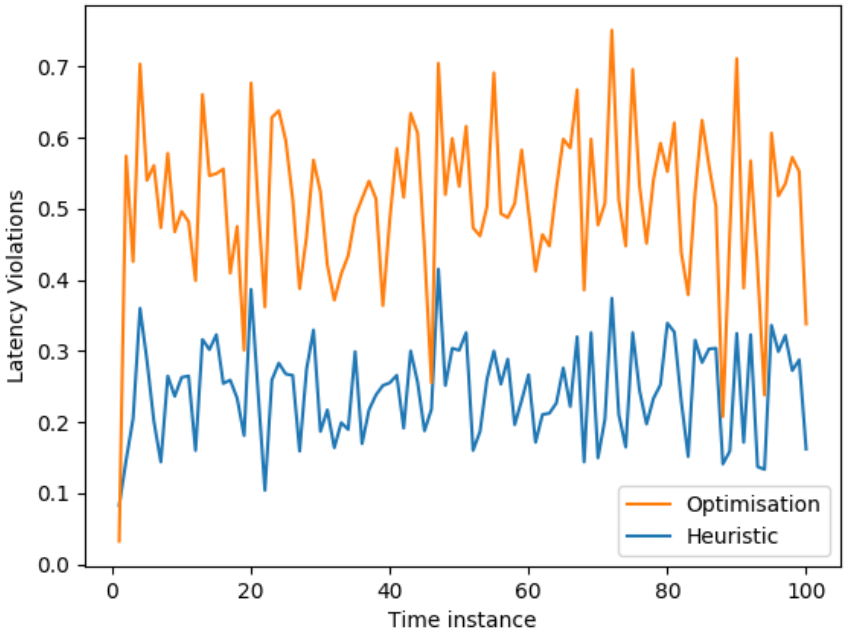

Figure 5.16: Latency violations per time instance

Gamma distribution with the characteristics described.

In Figure 5.17, we compare the resulting cumulative path latencies of the optimisation models proposed in Section 3.6 and the Nestor formulation. We average out the measurements of multiple experimental runs with the same mean latency distribution to obtain an indicator for the performance of the proposed solutions. We then compare the best-performing optimal formulation with the MPD algorithm in Figure 5.18. Although faced with adverse network conditions, the heuristic placement of VNFs in the network provides near-optimal results, with no significant deviation from the optimisation model used.

Furthermore, we look at the mean placement latencies and variation of the optimal formulations in Figure 5.19. Using this technique, the mean and average values for the latencies present close values, with no significant outliers detected in placement strategies. We observe that the variation of the Nestor strategy is lower with respect to the mean latency increases, while the Context-based schemes provide some unpredictability (with, e.g., lower variation in the path latency scale of 80-100ms, and higher in 1-50ms).

We analyse the MPD algorithm under similar circumstances in Figure 5.20. Between the heuristic MPD and Context-based solution we observe that the variation in placement (error) is not directly linked to the increases in the mean latency scaling. The proposed MPD formulation,

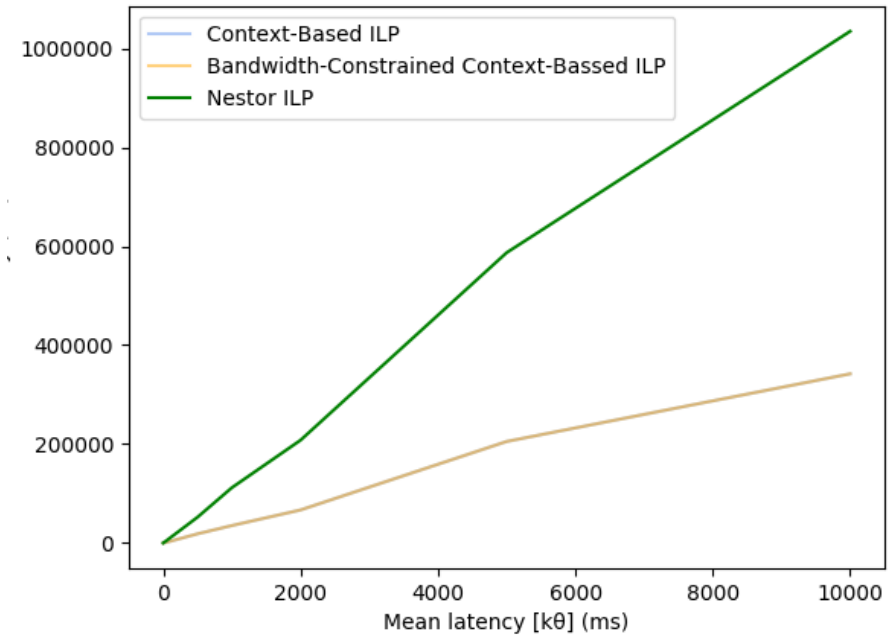

Figure 5.17: Optimal placement robustness

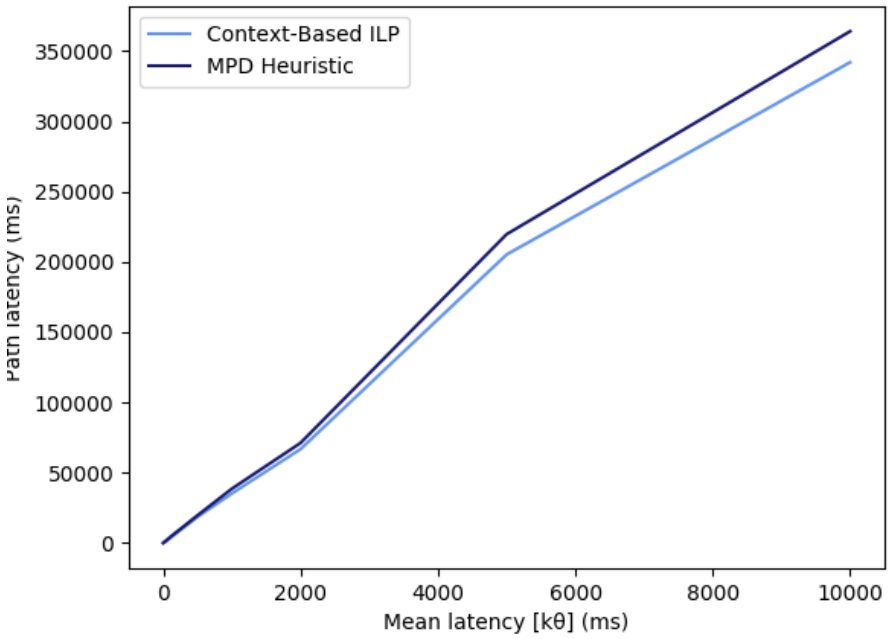

Figure 5.18: MPD robustness

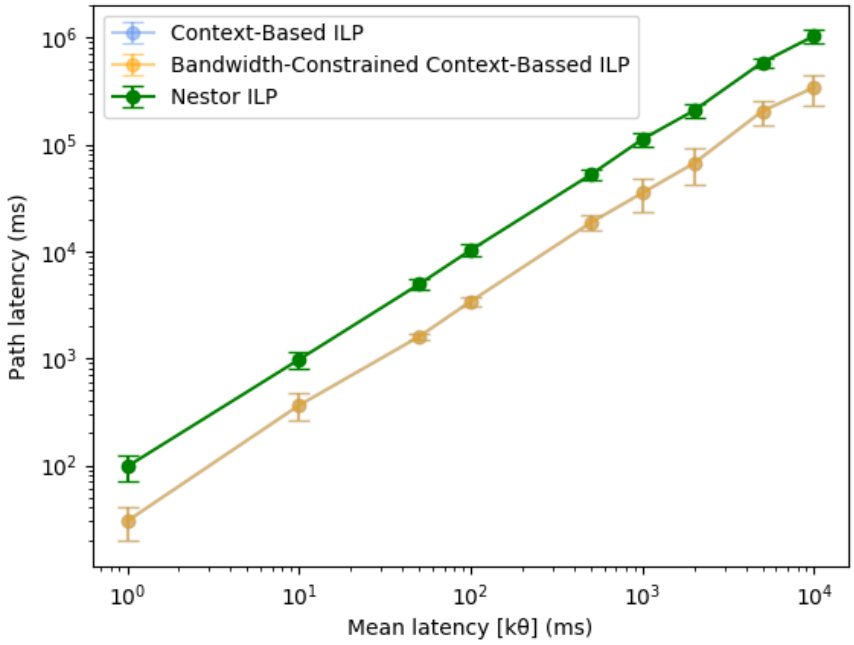

Figure 5.19: Mean placement latencies and errors of optimal formulations

although designed with edge network as its principal application domain, provides robust allocation when operating in infrastructures with larger link latencies, with convergence with the Optimal formulation starting to occur in the 100ms range (e.g., backbone).

5.5 Summary

This chapter has presented the important characteristics of the proposed lightweight NF architecture and associated management and orchestration framework, aimed at providing composable security services in resource-constrained networks.

It used performance measurements taken from real-world environments to compare the critical aspects of microservice NFs with the equivalent state-of-the-art monolithic implementations, showing equivalent or improved performance from architectural choice.

The key aspects of deployment on resource-constrained environments have been identified and evaluated, with the impact on network-level performance investigated. It has investigated the impact of on-the-fly generation of security VNF software artifacts and autonomous lifecycle management of the hosting infrastructure.

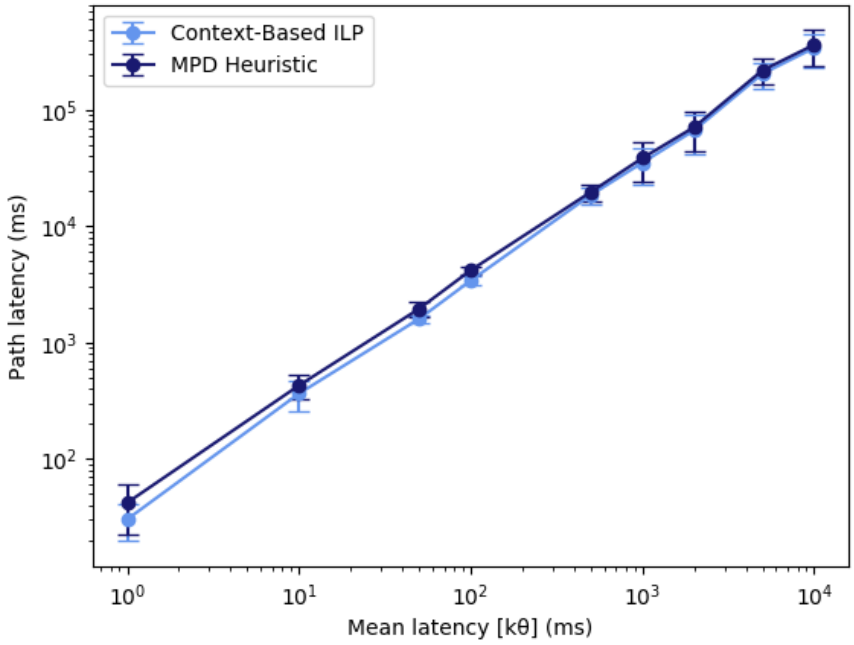

Figure 5.20: Mean placement latencies and errors of MPD

Finally, the performance of the heuristic Minimal Path Deviation allocation was compared against context-based and resource-consolidation optimal solutions over a simulated national infrastructure, having near-optimal performance in placement of complex security service chains under multiple infrastructure conditions, while allowing operators to maintain best practices.

Chapter 6

Conclusions and Future Research Directions

6.1 Overview

The final chapter highlights the contributions of the research performed in Section 6.2. In Section 6.3 the thesis statement is revisited in light of the research presented. The lessons learned throughout are presented in Section 6.4. Future research directions, including further use-cases for orchestration of microservice-based VNF architectures, integration with emerging research domains, and general directions for Software-Defined Security policies are highlighted in Section 6.5. Finally, Section 6.6 concludes the thesis.

6.2 Contributions

This work has applied the latest technologies and principles of network programmability, including NFV and SDN, and combined them with Context-based Security paradigms to bring network resilience services to the network edge. Through a novel framework for security network function orchestration operators are able to manage Virtual Network Functions in a heterogeneous network environment, maximising infrastructure resource usage. This capability allows the providers to offer new value-added services to end-users, improving Return on Investment by reducing capital and operational expenditures.

Through the benefits of on-path service chain placement, this work has demonstrated significant improvement of end-user Quality of Service. Residual network resource utilisation is maximised, the requirement for Data Centre deployments at the network edge is reduced, and critical network infrastructure can be protected against malicious or undesired usage.

By adoption of the microservice design pattern, network functions can be implemented in a modular, lightweight fashion. These network functions have reduced deployment time, lower packet processing overhead, and can operate in resource-constrained environments.

In Chapter 2 we have presented an overview of the evolution of network programmability, cybersecurity solutions, and summarised the existing state-of-the-art on the topic of Network Function Virtualisation from the perspective of a network operator applying security principles.

Chapter 3 analysed the limitations of previous network security paradigms and orchestration algorithms, and motivated the need for a modular, device-centric approach. It outlined the high-level requirements and design principles for next-generation network service deployment, compared different security architectures and paradigms, and presented a latency-optimal in-line service chain placement algorithm, expressed as an Integer Linear Programming Optimisation Problem.

The technical details of an implementation were presented in Chapter 4. A bottom-up framework was presented, detailing modular security services implementations, placement and lifecycle management strategies, and the network traffic in several deployment scenarios. It considered real-world constraints in providing allocation of complex service chains, and presented a heuristic Minimal Path Deviation Allocation algorithm that provides near-optimal security service chain placement for roaming clients.

Chapter 5 gave a comprehensive evaluation of the proposed framework. It highlighted the benefits of modular security services through performance benchmarks. It then analysed the responsiveness of the framework in environments with roaming clients which required rapid deployment of complex security service chains. Finally, an evaluation of the latency-optimal in-line service chain placement orchestration and its' heuristic approximation is conducted over a simulated network topology with real-world latency characteristics.

6.3 Thesis Statement Revisited

In this section, the thesis statement, first iterated in Section 1.2, is repeated, with the remainder of the section indicating how it has been addressed.

The thesis statement is reiterated, as follows:

The deployment of VNFs introduces flexibility and dynamism in response to the increased demand for network-enhanced services in modern and next-generation networks. Network operators' enforcement of cyber-security policies needs to respond to the temporal variations within the network, while providing service-level agreements to end-users. This work asserts that creating, dynamically managing, and monitoring lightweight security modules in edge networks (for example public wireless LANs, 5G cell clusters, Autonomous Vehicle networks) will allow operators to provide assurances on device-to-device communication. The work focuses on network functions that, through behaviour and placement within the network, lead to low traffic latency overhead in edge network paths. The flexible nature of security best practices requires that the work not limit the expression of security service composition and placement.

The thesis has started by describing the need for unified network management and introducing important underlying achievements such as SDN and NFV. It motivated the aims of next-generation networks providing low-latency services for users, and showed limitations of existing research and industry solutions. It analysed the network operator's requirements for providing cybersecurity services, and identified the existing state-of-the-art in using virtualised security functions.

To provide security services in close proximity to the end users (for example at the network edge), the concept of lightweight, targeted, microservice-based security functions has been evaluated. As shown, lightweight services can be created easily and hosted on various low-cost devices (e.g., home routers, or IoT devices). It highlighted a microservice-based paradigm for services to rapidly initialise, or be reconfigured in a matter of seconds, instead of minutes. Several real-world examples of security functions have been presented in this thesis.

For operation in highly dynamic networks, where users join and leave the network frequently, several management strategies have been investigated. As evidenced in our evaluation, performing autonomous lifecycle management can reduce the complexity in orchestrating multiple VNFs in large networks. Furthermore, creation of targeted microservices for security and their on-demand deployment based on user behaviour has been proven to provide customisable and transparent network services.

Balancing low-latency network connectivity with computationally complex placement of VNF chains is crucial when considering temporal elements of networks (e.g., user locations, latency variability). The trade-off between these two aspects has been analysed, with a solution using Integer Linear Programming for latency-optimal Service Chain placement, and an associated heuristic that provides near-optimal allocation while providing faster orchestration of complex network services. These solutions have been compared against state-of-the-art solutions that aim on minimising operator costs. Results have shown that, Service Level Agreements are met when employing a strategy that enables the minimisation of latency and data paths.

This thesis has demonstrated how microservice-based VNFs can be created, deployed, managed and orchestrated in resource-constrained network environments to provide targeted, customisable, and transparent complex infrastructure security services for end-users. It builds upon a well known container management framework, enhancing its capability to operate as an NFV platform through improved service deployment, placement, and lifecycle management.

6.4 Lessons Learned

This thesis is a result of a research endeavour spanning multiple years. Over time, several important lessons have been learned and multiple obstacles have been overcome. In this Section we cover note some of the main achievements of the work, and some important lessons learned.

6.4.1 Research Contribution in Systems

Research within the wider domain of Computer Systems (e.g., Operating Systems, Distributed Systems, Networks, Parallelism, etc.) is often challenging, proposing novel ideas often difficult. Contributions build upon a body of knowledge that has been constructed over a decades-long process. Some ideas presented within a field can also be applied to other knowledge domains and environments, such as the problem of task scheduling in many-core systems being adapted for Virtual Machine placement within a Data Centre.

Innovation relies on both the novel concepts as well the methods used and resulting implementation. Many of the state-of-the-art solutions that have gathered wide recognition in both academic and industrial domains are accompanied by thorough methodology and implementation, from Google's Borg project [143] underpinning many of the current deployments of distributed computation including traditional web services and virtual network functions, to the Google File

System [53] that put forward the main principles of modern large-scale distributed computation, to the Ethane [22] and OpenFlow [94] projects that drive evolution of modern networks.

6.4.2 Reality of Current Networks

Computer Networks have frequently faced the problem of ossification. Development and adoption of new ideas is delayed because of widespread adoption of current solutions. For example, the concept of Active Networking [135], which looked at an increasingly programmable network, has failed to gain the network traction because the required changes were not compatible with existing network devices. The adoption of OpenFlow also faced some challenges because of autonomous network protocols, such as BGP, would not function properly. Support in adoption of new technologies in current environments, such as adapting existing switches for OpenFlow support [29], is crucial for the adoption of new paradigms and the transition towards increasingly flexible and programmable networks.

Computer Networks also become increasingly complicated and difficult to model. The number of devices, applications, behaviours, and protocols makes accurate recreations without reliable information. Sometimes, a restricted view, with justified assumptions on network behaviour, is required. For example, network traffic in modern Data Centres is assumed to be predominantly east-west, because the majority of the workload involves data processing solutions. Similarly, this thesis assumes that edge networks are heterogeneous, motivated by the prevalence of commodity servers which are based on the x86 instruction set, and also the recent popularity of low-cost ARM devices that can run general-purpose software [140].

6.4.3 Security in Computer Networks

The topic of security within computer networks is extremely broad, and can have different meanings depending on the context within which it is presented. It encompasses broad topics, from vulnerability analysis, exploitation vectors, to privacy, encryption, trust, to operator isolation, infrastructure resilience, and large-scale malicious usage.

This thesis has focused on tackling the challenges of provisioning of security in an emerging network topology. By exploiting state-of-the-art advancements in network softwarisation, programmability, and virtualisation, a deployment model which aims to better utilise residual computational capacity within the network in an efficient way, without impacting end-user ex-

perience in the process.

In achieving this, one of the biggest challenges was accurately describing the security deployments that are in use in real-world environments. Operators often treat this information as confidential, as it could be used to circumvent protection mechanisms. Information is pieced together from publications, incident postmortem reports, marketing material of various solutions.

But some information becomes quickly outdated. Evolution of dataplane programmability solutions, such as the P4 language [17] and associated Barefoot/Intel Tofino hardware implementation that makes it possible to deploy within production networks, opens a gateway towards a new generation of network deployments. The traditional DNS Amplification DDoS attack could be prevented in real-time by dataplane solutions that operate at line rate within the network. This (potential) evolution would not invalidate the work done so far, but requires that existing contributions remain forward-looking: wider integration within the network ecosystem, that encourages for further improvement, and builds upon previous work.

6.5 Future Research Directions

6.5.1 Further Use-Cases

The work presented in this thesis has been channelled towards existing edge network architectures (e.g., residential and public networks, IoT). Emerging network architectures that exhibit similar characteristics can benefit from this approach, with specific improvements related to their specific characteristics. For example, next-generation cellular networks can use spatial-awareness to predict user locations and prepare migration of network services to minimise downtime.

Furthermore, certain scenarios allow for the possibility of hosting VNFs on the network client device itself. One such example use-case is in autonomous vehicle networks, in which devices benefit from specialised processing hardware and available resources for execution of local functionality. Management and orchestration in these conditions brings special energy-efficiency concerns, which pose an interesting research challenge.

Finally, the principles of this thesis can be applied to Data Centre environments. With an established infrastructure relying on SDN and NFV, adoption of lightweight VNFs can provide

increased operator resource utilisation, and reduce capital and operational expenditure related to the management of the infrastructure. From a resilience perspective, the work highlighted in this thesis can be used to rapidly detect and mitigate emerging infrastructure attacks that are commonly targeting DCs. With Cloudflare reporting[1] that DDoS attacks are increasing in frequency and volume, operators are required to rapidly adapt to emerging threats.

6.5.2 Programmable Data Planes

With Intel's acquisition of Altera in 2016, and Barefoot Networks in 2019, deployment of data plane programmable NFs is likely to become ubiquitous in the near future. Network services can benefit from emerging data plane programmability in implementing functionality and VNF telemetry. Use of "bump-in-the-wire" functions on programmable network interfaces and switches is becoming an emerging area of network research, and data plane network telemetry is becoming of increased interest for commercial operators.

Similarly, advancements in hardware offloading and virtualisation, paves they way for advanced dataplane services for regular applications. nVidia Mellanox interfaces[2] provide support for eXpress Data Path[3] offloading. In the near future, operators can leverage this integration, along with hardware virtualisation features (e.g., SR-IOV, VT-c, and VT-d) to significantly reduce the overheads of virtualisation, provide transparent network functions before packets even leave the host, and present new use-cases for the network.

6.5.3 Service Request Formalisation

To the best of our knowledge, there is no formal standardisation for the expression of Service Requests. Each NFV platform, including the one presented within this dissertations, makes assumptions on the types of services, how chaining is achieved, and definition requirements and limitations. In order for continuous innovation to occur within the NFV domain, a formalisation of the chaining behaviour, steering considerations, and system description is required. This would enable real-world applicability for multiple-operator VNF compatibility, and multiple domain NFV management, and better understanding of network behaviour, enabling further development of NFV platforms.

[1] https://blog.cloudflare.com/network-layer-ddos-attack-trends-for-q2-2020/ Retrieved November 2020

[2] https://developer.nvidia.com/blog/accelerating-with-xdp-over-mellanox-connectx-nics/

[3] https://www.redhat.com/en/blog/capturing-network-traffic-express-data-path-xdp-environment Retrieved September 2021

6.5.4 Software-Defined Security

The network security paradigms have started shifting, leveraging modern network management solutions, such as SDN and NFV. Yet, the vision of user-centric network security requires further sustained efforts from academia and industry to become a tangible reality.

The scope of this thesis has allowed exploration of how network services can be designed, and how management and orchestration could be performed in order to provide transparent, responsive, and resource-efficient service delivery. However, other areas for providing flexible security services aimed to improve infrastructure resilience provide promising research directions. Foremost is the classification mechanism used to determine user context and associated service requirements. The work in this area currently relies on operator-defined device classification. Research into dynamic, run-time classification of devices and application traffic can expand the flexibility of any context-based framework. Early approaches based on machine learning or artificial intelligence for traffic classification could be adapted to provide promising results for widespread adoption of context-based security.

Furthermore, alert aggregation schemes for network-wide event monitoring can be used to improve and preempt future infrastructure attacks. To this end, providing unified, extensible, and clear communication mechanisms between services deployed throughout a network such that it can dynamically react to emerging network events is paramount. Context-based frameworks are fundamentally based on the transition towards increasingly stricter security services as infrastructure threats evolve, and further research avenues open up for investigation regarding this area.

Alongside these research efforts, focus on improved reliability and fault tolerance of VNFs providing critical functionality is required. Since security services are often the initial target of infrastructure attacks, mechanisms to detect resource contention and methods for mitigation need to be considered. By leveraging network telemetry for latency analysis of VNF performance, threats as those mentioned above can be further identified and mitigated.

6.6 Concluding Remarks

The computer networking industry is currently evolving towards enhanced programmability. Continuous improvements in network virtualisation and softwarisation enable automated, zero-touch management of the infrastructure with a clear separation of layers, from the physical

up to the application. This thesis attempted to unify the concepts of network softwarisation and infrastructure security, allowing for deployment of flexible services in dynamic network environments which are becoming ubiquitous.

This work confirmed the applicability of the context-based security paradigms that have been proposed in the literature and outlined the essential concepts of an NFV framework and relevant VNFs that can be used in resource-constrained environments. It shed light into new research directions for core aspects of the aforementioned paradigms that improve applicability, and ushers in the era of software-defined security for next-generation networks.

Publications

The work reported in this thesis has led to the following publications:

- Mircea M. Iordache-Sica, Christos Anagnostopoulos, and Dimitrios P. Pezaros. Towards qos-aware provisioning of chained virtual security services in edge networks. In *2021 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2021

Bibliography

[1] Yehuda Afek, Anat Bremler-Barr, Yotam Harchol, David Hay, and Yaron Koral. Mca 2: multi-core architecture for mitigating complexity attacks. In *2012 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 235–246. IEEE, 2012.

[2] Yehuda Afek, Anat Bremler-Barr, Yotam Harchol, David Hay, and Yaron Koral. Making DPI Engines Resilient to Algorithmic Complexity Attacks. *IEEE/ACM Transactions on Networking*, 24(6):3262–3275, 2016.

[3] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[4] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. *Proceedings of NSDI 2010: 7th USENIX Symposium on Networked Systems Design and Implementation*, pages 281–295, 2010.

[5] Abeer Ali, Christos Anagnostopoulos, and Dimitrios P Pezaros. Resource-aware placement of softwarised security services in cloud data centers. In *2017 13th International Conference on Network and Service Management (CNSM)*, pages 1–5. IEEE, 2017.

[6] Abeer Ali, Richard Cziva, Simon Jouet, and Dimitrios P Pezaros. SDNFV-based DDoS detection and remediation in multi-tenant , virtualized infrastructures. In *Guide to Security in SDN and NFV: Challenges, Opportunities, and Applications*, pages 1–27. Springer, 2017.

[7] Jianping An, Kai Yang, Jinsong Wu, Neng Ye, Song Guo, and Zhifang Liao. Achieving Sustainable Ultra-Dense Heterogeneous Networks for 5G. *IEEE Communications Magazine*, 55(12):84–90, dec 2017.

[8] Bilal Anwer, Theophilus Benson, Nick Feamster, Dave Levin, and Jennifer Rexford. A slick control plane for network middleboxes. *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, page 147, 2013.

[9] Paul Barford and David Plonka. Characteristics of network traffic flow anomalies. In *Proceedings of the First ACM SIGCOMM Workshop on Internet Measurement - IMW '01*, page 69, New York, New York, USA, 2004. ACM Press.

[10] Md Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, and Raouf Boutaba. PolicyCop: An Autonomic QoS Policy Enforcement Framework for Software Defined Networks. In *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, pages 1–7. IEEE, nov 2013.

[11] Md Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, Raouf Boutaba, and Otto Carlos Muniz Bandeira Duarte. Orchestrating Virtualized Network Functions. *IEEE Transactions on Network and Service Management*, 13(4):725–739, 2016.

[12] Cataldo Basile, Christian Pitscheider, Fulvio Risso, Fulvio Valenza, and Marco Vallini. Towards the Dynamic Provision of Virtualized Security Services. In *Communications in Computer and Information Science*, volume 470, pages 65–76. Springer, 2015.

[13] Andreas Baumgartner, Varun S. Reddy, and Thomas Bauschert. Mobile core network virtualization: A model for combined virtual core network function placement and topology optimization. *1st IEEE Conference on Network Softwarization: Software-Defined Infrastructures for Networks, Clouds, IoT and Services, NETSOFT 2015*, 2015.

[14] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, and Bob Lantz. ONOS: towards an open, distributed SDN OS. *Proceedings of the third workshop on Hot topics in software defined networking - HotSDN '14*, pages 1–6, 2014.

[15] Mark Berman, Jeffrey S. Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. GENI: A federated testbed for innovative network experiments. *Computer Networks*, 61(2014):5–23, 2014.

[16] Pat Bosshart, Dan Daly, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. Programming Protocol-Independent Packet Processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2013.

[17] Pat Bosshart, Dan Daly, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. Program-

ming Protocol-Independent Packet Processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2013.

[18] Patrick Brézillon and Ghita Kouadri Mostéfaoui. Context-based security policies: A new modeling approach. *Proceedings - Second IEEE Annual Conference on Pervasive Computing and Communications, Workshops, PerCom*, pages 154–158, 2004.

[19] Matthew Caesar, Donald Caldwell, Aman Shaikh, Nick Feamster, Jennifer Rexford, and Jacobus van der Merwe. Design and implementation of a routing control platform. *2nd Symposium on Networked Systems Design and Implementation, NSDI 2005*, pages 15–28, 2005.

[20] Giuseppe C. Calafiore and Laurent El Ghaoui. *Optimization Models*. Cambridge university press, 2014.

[21] Ken Calvert. Reflections on network architecture. *ACM SIGCOMM Computer Communication Review*, 36(2):27–30, 2006.

[22] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane. *ACM SIGCOMM Computer Communication Review*, 37(4):1–12, 2007.

[23] Martin Casado, Tal Garfinkel, Aditya Akella, Michael J. Freedman, Dan Boneh, Nick McKeown, and Scott Shenker. SANE: A protection architecture for enterprise networks. *15th USENIX Security Symposium*, 49(50):137–151, 2006.

[24] Lee Chao. vCloud Director. In *Virtualization and Private Cloud with VMware Cloud Suite*, pages 355–404. Routledge, mar 2017.

[25] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical Report 2.1, Dartmouth College, 2000.

[26] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. PlanetLab: An overlay testbed for broad-coverage services. *Computer Communication Review*, 33(3):3–12, 2003.

[27] Byonggon Chun, Jihun Ha, Sewon Oh, Hyunsung Cho, and MyeongGi Jeong. Kubernetes Enhancement for 5G NFV Infrastructure. In *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1327–1329. IEEE, oct 2019.

[28] Mark Claypool and Kajal Claypool. Latency and player actions in online games. *Communications of the ACM*, 49(11):40, nov 2006.

[29] Levente Csikor, Márk Szalay, Gábor Rétvári, Gergely Pongrácz, Dimitrios P Pezaros, and László Toka. Transition to sdn is harmless: Hybrid architecture for migrating legacy ethernet switches to sdn. *IEEE/ACM Transactions On Networking*, 28(1):275–288, 2020.

[30] Lin Cui, Fung Po Tso, and Weijia Jia. Enforcing network policy in heterogeneous network function box environment. *Computer Networks*, 138:108–118, 2018.

[31] Lin Cui, Fung Po Tso, Dimitrios Pezaros, Weijia Jia, and Wei Zhao. PLAN: Joint policy- and network-aware VM management for cloud data centers. *IEEE Transactions on Parallel and Distributed Systems*, PP(99), 2016.

[32] Richard Cziva, Christos Anagnostopoulos, and Dimitrios P. Pezaros. Dynamic, Latency-Optimal vNF Placement at the Network Edge. *Proceedings - IEEE INFOCOM*, 2018-April:693–701, 2018.

[33] Richard Cziva, Simon Jouët, David Stapleton, Fung Po Tso, and Dimitrios P. Pezaros. SDN-Based Virtual Machine Management for Cloud Data Centers. *IEEE Transactions on Network and Service Management*, 13(2):212–225, 2016.

[34] Richard Cziva, Simon Jouet, Kyle J.S. White, and Dimitrios P. Pezaros. Container-based network function virtualization for software-defined networks. *Proceedings - IEEE Symposium on Computers and Communications*, 2016-Febru:415–420, 2016.

[35] Richard Cziva, Simon Jouet, Kyle J.S. White, and Dimitrios P. Pezaros. Container-based network function virtualization for software-defined networks. *Proceedings - IEEE Symposium on Computers and Communications*, 2016-Febru:415–420, 2016.

[36] Richard Cziva, Christopher Lorier, and Dimitrios P. Pezaros. Ruru: High-speed, flow-level latency measurement and visualization of live internet traffic. *SIGCOMM Posters and Demos 2017 - Proceedings of the 2017 SIGCOMM Posters and Demos, Part of SIGCOMM 2017*, pages 46–47, 2017.

[37] Richard Cziva and Dimitrios P. Pezaros. Container Network Functions: Bringing NFV to the Network Edge. *IEEE Communications Magazine*, 55(6):24–31, 2017.

[38] David Day and Benjamin Burns. A Performance Analysis of Snort and Suricata Network Intrusion Detection and Prevention Engines. *ICDS 2011, The Fifth International Conference on Digital Society*, (c):187–192, 2011.

[39] David Dietrich, Ahmed Abujoda, Amr Rizk, and Panagiotis Papadimitriou. Multi-provider service chain embedding with nestor. *IEEE Transactions on Network and Service Management*, 14(1):91–105, 2017.

[40] R. Doriguzzi-Corin, S. Scott-Hayward, D. Siracusa, and E. Salvadori. Application-centric provisioning of virtual security network functions. *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN 2017*, 2017-Janua:276–279, 2017.

[41] Roberto Doriguzzi-Corin, Sandra Scott-Hayward, Domenico Siracusa, Marco Savi, and Elio Salvadori. Dynamic and Application-Aware Provisioning of Chained Virtual Security Network Functions. *IEEE Transactions on Network and Service Management*, 17(1):294–307, 2020.

[42] ETSI. Network Functions Virtualisation,An Introduction, Benefits, Enablers, Challenges & Call for Action. Technical report, European Telecommunications Standards Institute, 2012.

[43] Florian Fainelli. The OpenWrt embedded development framework. *Proceedings of the Free and Open Source Software Developers European Meeting*, 2008.

[44] Lyndon Fawcett, Sandra Scott-Hayward, Matthew Broadbent, Andrew Wright, and Nicholas Race. Tennison: A distributed SDN framework for scalable network security. *IEEE Journal on Selected Areas in Communications*, 36(12):2805–2818, 2018.

[45] Seyed Kaveh Fayazbakhsh, Luis Chiang, Vyas Sekar, Minlan Yu, and Jeffrey C Mogul. Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags. *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2014)*, pages 543–546, 2014.

[46] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to SDN: An intellectual history of programmable networks. *Computer Communication Review*, 44(2):87–98, 2014.

[47] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann De Meer, and Xavier Hesselbach. Virtual Network Embedding : A Survey. *IEEE Communications Surveys & Tutorials*, 15(4):1888–1906, 2013.

[48] Linux Foundation. Data plane development kit, 2021.

[49] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez, and E. Vazquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1):18–28, 2009.

[50] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers and Security*, 28(1-2):18–28, 2009.

[51] Anastasius Gavras, Arto Karila, Serge Fdida, Martin May, and Martin Potts. Future internet research and experimentation: The FIRE initiative. *Computer Communication Review*, 37(3):89–92, 2007.

[52] MacIej Gawel and Krzysztof Zielinski. Analysis and Evaluation of Kubernetes Based NFV Management and Orchestration. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 511–513. IEEE, jul 2019.

[53] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, 2003.

[54] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[55] Abhishek Gupta, M. Farhan Habib, Pulak Chowdhury, Massimo Tornatore, and Biswanath Mukherjee. On service chaining using Virtual Network Functions in Network-enabled Cloud systems. *International Symposium on Advanced Networks and Telecommunication Systems, ANTS*, 2016-February:1–3, 2016.

[56] B. B. Gupta and Omkar P. Badve. Taxonomy of DoS and DDoS attacks and desirable defense mechanism in a Cloud computing environment. *Neural Computing and Applications*, 28(12):3655–3682, 2017.

[57] Optimization Gurobi. Gurobi Optimizer Reference Manual, Version 5.0. *Www.Gurobi.Com*, 2018.

[58] W. Haeffner, J. Napper, M. Stiemerling, D. Lopez, and J. Uttaro. Service Function Chaining Use Cases in Mobile Networks. Technical Report draft-aranda-sfc-dp-mobile-01, IETF, 2016.

[59] J. Halpern and C Pignataro. Service Function Chaining (SFC) Architecture. *IETF RFC 7665*, 84:1–32, 2015.

[60] Sangjin Han, Keon Jang, Aurojit Panda, Shoumik Palkar, Dongsu Han, and Sylvia Ratnasamy. SoftNIC: A Software NIC to Augment Hardware. *TechReport*, 2015.

[61] Brandon Heller. OpenFlow Switch Specification 1.0.0. *Current*, 0:1–36, 2009.

[62] Rhys Hill, Christopher Madden, Anton Van Den Hengel, Henry Detmold, and Anthony Dick. Measuring latency for video surveillance systems. *DICTA 2009 - Digital Image Computing: Techniques and Applications*, pages 89–95, 2009.

[63] Michio Honda, Felipe Huici, Giuseppe Lettieri, and Luigi Rizzo. mSwitch. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research - SOSR '15*, volume 10, pages 1–13, New York, New York, USA, 2015. ACM Press.

[64] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing—A key technology towards 5G. ETSI White Paper. *ETSI White Paper*, 11(11):1—-16., 2015.

[65] IEEE. Ieee standard for local and metropolitan area networks - station and media access control connectivity discovery. *IEEE Std 802.1AB-2016 (Revision of IEEE Std 802.1AB-2009)*, pages 1–146, 2016.

[66] Kentaro Imagane, Kenji Kanai, Jiro Katto, Toshitaka Tsuda, and Hidenori Nakazato. Performance evaluations of multimedia service function chaining in edge clouds. In *CCNC 2018 - 2018 15th IEEE Annual Consumer Communications and Networking Conference*, volume 2018-Janua, pages 1–4. IEEE, jan 2018.

[67] John Ioannidis and SM Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Network and Distributed System Security Symposium '02*, pages 1–12, 2002.

[68] Mircea M. Iordache-Sica, Christos Anagnostopoulos, and Dimitrios P. Pezaros. Towards qos-aware provisioning of chained virtual security services in edge networks. In *2021 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2021.

[69] Narjes Tahghigh Jahromi, Roch H Glitho, Adel Larabi, and Richard Brunner. An nfv and microservice based architecture for on-the-fly component provisioning in content delivery networks. In *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–7. IEEE, 2018.

[70] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jonathan Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, sep 2013.

[71] Insun Jang, Sukjin Choo, Myeongsu Kim, Sangheon Pack, and Myung Ki Shin. Optimal network resource utilization in service function chaining. *IEEE NETSOFT 2016 - 2016 IEEE NetSoft Conference and Workshops: Software-Defined Infrastructure for Networks, Clouds, IoT and Services*, pages 11–14, 2016.

[72] Joe Wenjie Jiang, Tian Lan, and Minghua Chen. Joint VM Placement and Routing for Data Center Traffic Engineering. *2012 Proceedings IEEE INFOCOM*, pages 2876–2880, 2012.

[73] Wolfgang John, Konstantinos Pentikousis, George Agapiou, Eduardo Jacob, Mario Kind, Antonio Manzalini, Fulvio Risso, Dimitri Staessens, Rebecca Steinert, and Catalin Meirosu. Research Directions in Network Service Chaining. In *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, pages 1–7. IEEE, nov 2013.

[74] Simon Jouet and Dimitrios P Pezaros. BPFabric : Data Plane Programmability for Software Defined Networks. In *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, pages 38–48, 2017.

[75] Georgios P. Katsikas, Tom Barbette, Dejan Kostić, Rebecca Steinert, and Gerald Q. Maguire. Metron: NFV service chains at the true speed of the underlying hardware. In *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018*, 2018.

[76] Georgios P. Katsikas, Marcel Enguehard, Maciej Kuźniar, Gerald Q. Maguire Jr, and Dejan Kostić. SNF: synthesizing high performance NFV service chains. *PeerJ Computer Science*, 2:e98, 2016.

[77] Steve Klabnik and Carol Nichols. *The Rust Programming Language (Covers Rust 2018)*. No Starch Press, 2019.

[78] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 2000.

[79] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. DDoS in the IoT: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[80] Tung-Wei Kuo, Bang-Heng Liou, Kate Ching-Ju Lin, and Ming-Jer Tsai. Deploying chains of virtual network functions: On the relation between link and server usage. *IEEE/ACM Transactions On Networking*, 26(4):1562–1576, 2018.

[81] Ar Kar Kyaw, Yuzhu Chen, and Justin Joseph. Pi-IDS: Evaluation of open-source intrusion detection systems on Raspberry Pi 2. *2015 2nd International Conference on Information Security and Cyber Forensics, InfoSec 2015*, pages 165–170, 2016.

[82] Rafael Laufer, Massimo Gallo, Diego Perino, and Anandatirtha Nandugudi. CliMB. *ACM SIGCOMM Computer Communication Review*, 46(4):17–22, dec 2016.

[83] E. L. Lawler and D. E. Wood. Branch-and-Bound Methods: A Survey. *Operations Research*, 14(4):699–719, aug 1966.

[84] Guanglei Li, Huachun Zhou, Bohao Feng, and Guanwen Li. Context-aware service function chaining and its cost-effective orchestration in multi-domain networks. *IEEE Access*, 6:34976–34991, 2018.

[85] Hongda Li, Hongxin Hu, Guofei Gu, Gail-Joon Ahn, and Fuqiang Zhang. vNIDS. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security - CCS '18*, volume v, pages 17–34, New York, New York, USA, 2018. ACM Press.

[86] A. Lombardo, A. Manzalini, G. Schembra, G. Faraci, C. Rametta, and V. Riccobene. An open framework to enable NetFATE (Network Functions at the edge). *1st IEEE Conference on Network Softwarization: Software-Defined Infrastructures for Networks, Clouds, IoT and Services, NETSOFT 2015*, 2015.

[87] D R Lopez. OpenMANO: The dataplane ready open source NFV MANO stack. In *IETF Meeting Proceedings, Dallas, Texas, USA*, 2015.

[88] David G Luenberger and Yinyu Ye. *Linear and Nonlinear Programming*, volume 228 of *International Series in Operations Research & Management Science*. Springer International Publishing, Cham, 2016.

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

[90] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. Fog Computing: A taxonomy, survey and future directions. *Internet of Things*, 0(9789811058608):103–130, 2018.

[91] Joao Martins, Mohamed Ahmed, Costin Raiciu, and Felipe Huici. Enabling fast, dynamic network processing with clickOS. *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, page 67, 2013.

[92] Manuel Mazzara and Betrand Meyer. *Present and Ulterior Software Engineering*. Springer International Publishing, Cham, 2017.

[93] Steven Mccanne and Van Jacobson. The BSD Packet Filter : A New Architecture for User-level Packet Capture. *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings*, pages 1–11, 1993.

[94] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, mar 2008.

[95] Ahmed M. Medhat, Tarik Taleb, Asma Elmangoush, Giuseppe A. Carella, Stefan Covaci, and Thomas Magedanz. Service Function Chaining in Next Generation Networks: State of the Art and Research Challenges. *IEEE Communications Magazine*, 55(2):216–223, 2017.

[96] Jan Medved, Robert Varga, Anton Tkacik, and Ken Gray. OpenDaylight: Towards a model-driven SDN controller architecture. *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014, WoWMoM 2014*, 2014.

[97] Gerhard Miinz and Georg Carle. Real-time Analysis of Data Attack Detection Flow Network. *Integrated Network Management*, pages 100–108, 2007.

[98] Rashid Mijumbi, Joan Serrat, Juan Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys and Tutorials*, 18(1):236–262, 2016.

[99] Abdi R. Modarressi and Ronald A. Skoog. Signaling system No. 7: A tutorial. *IEEE Communications Magazine*, 28(7):19–20, 22, 1990.

[100] Hendrik Moens and Filip De Turck. VNF-P: A model for efficient placement of virtualized network functions. *Proceedings of the 10th International Conference on Network and Service Management, CNSM 2014*, pages 418–423, 2014.

[101] Adam Morrison, Lei Xue, Ang Chen, and Xiapu Luo. Enforcing Context-Aware BYOD Policies with In-Network Security. *10th USENIX Workshop on Hot Topics in Cloud Computing*, 2018.

[102] Ghita Kouadri Mostéfaoui and Patrick Brézillon. Modeling context-based security policies with contextual graphs. *Proceedings - Second IEEE Annual Conference on Pervasive Computing and Communications, Workshops, PerCom*, pages 28–32, 2004.

[103] Katta G Murty and Feng-Tien Yu. *LINEAR COMPLEMENTARITY , LINEAR AND NONLINEAR PROGRAMMING*. Helderman-Verlag, 1988.

[104] Balázs Németh, János Czentye, Gábor Vaszkun, Levente Csikor, and Balázs Sonkoly. Customizable real-time service graph mapping algorithm in carrier grade networks. *2015 IEEE Conference on Network Function Virtualization and Software Defined Network, NFV-SDN 2015*, pages 28–30, 2016.

[105] Eiji Oki. GLPK (GNU Linear Programming Kit). In *Linear Programming and Algorithms for Communication Networks*. 2012.

[106] Open Information Security Foundation. Suricata. *Suricata*, 2010.

[107] Jason Orlosky, Kiyoshi Kiyokawa, and Haruo Takemura. Virtual and Augmented Reality on the 5G Highway. *Journal of Information Processing*, 25(0):133–141, 2017.

[108] Angelos Pentelas, George Papathanail, Ioakeim Fotoglou, and Panagiotis Papadimitriou. Network service embedding across multiple resource dimensions. *IEEE Transactions on Network and Service Management*, 18(1):209–223, 2020.

[109] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J. Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Jonathan Stringer, Pravin Shelar, Keith Amidon, and Martín Casado. The design and implementation of open vSwitch. *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2015*, pages 117–130, 2015.

[110] Chuan Pham, Nguyen H Tran, Shaolei Ren, Walid Saad, and Choong Seon Hong. Traffic-aware and energy-efficient vnf placement for service chaining: Joint sampling and matching approach. *IEEE Transactions on Services Computing*, 13(1):172–185, 2017.

[111] Pekka Pirinen. A Brief Overview of 5G Research Activities. *Proceedings of the 1st International Conference on 5G for Ubiquitous Connectivity*, (February):17–22, 2014.

[112] David C. Plummer. An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 826, November 1982.

[113] Patrick Prosser. Exact algorithms for maximum clique: A computational study. *Algorithms*, 5(4):545–587, 2012.

[114] Long Qu, Chadi Assi, and Khaled Shaban. Delay-aware scheduling and resource optimization with network function virtualization. *IEEE Transactions on communications*, 64(9):3746–3758, 2016.

[115] P Quinn, J. Guichard, S. Kumar, M. Smith, W. Henderickx, T. Nadeau, P. Agarwal, R. Manur, A. Chauhan, J. Halpern, S. Majee, U. Elzur, D. Melman, P. Garg, B. McConnell, C. Wright, K. Glavin, C. Zhang, L. Fourie, R. Parker, and M Zarny. Network Service Header, 2014.

[116] Jürgen Quittek, Prashant Bauskar, Tayeb BenMeriem, Andy Bennett, Michel Besson, and Al Et. Network Functions Virtualisation (NFV); Management and Orchestration. *Gs Nfv-Man 001 V1.1.1*, 1:1–184, 2014.

[117] Jordi Ferrer Riera, Josep Batalle, Jose Bonnet, Miguel Dias, Michael McGrath, Giuseppe Petralia, Francesco Liberati, Alessandro Giuseppi, Antonio Pietrabissa, Alberto Ceselli, Alessandro Petrini, Marco Trubian, Panagiotis Papadimitrou, David Dietrich, Aurora

Ramos, Javier Melian, George Xilouris, Akis Kourtis, Tasos Kourtis, and Evangelos K. Markakis. TeNOR: Steps towards an orchestration platform for multi-PoP NFV deployment. *IEEE NETSOFT 2016 - 2016 IEEE NetSoft Conference and Workshops: Software-Defined Infrastructure for Networks, Clouds, IoT and Services*, pages 243–250, 2016.

[118] Roberto Riggio, Abbas Bradai, Tinku Rasheed, Julius Schulz-Zander, Slawomir Kuklinski, and Toufik Ahmed. Virtual network functions orchestration in wireless networks. *Proceedings of the 11th International Conference on Network and Service Management, CNSM 2015*, pages 108–116, 2015.

[119] Luigi Rizzo. NetMap: A novel framework for fast packet I/O. *Proceedings of the 2012 USENIX Annual Technical Conference, USENIX ATC 2012*, (257422):101–112, 2012.

[120] Lawrence G. Roberts. The ARPANET & computer networks. In *ACM Conference on the History of Personal Workstations, HPW 1986 - Conference Proceedings*, 1986.

[121] M Roesch. Snort: Lightweight Intrusion Detection for Networks. *LISA '99: 13th Systems Administration Conference*, pages 229–238, 1999.

[122] Eric C. Rosen. Vulnerabilities of network control protocols. *ACM SIGCOMM Computer Communication Review*, 1981.

[123] Rusty Russel. iptables(8), 1998.

[124] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)*, 1984.

[125] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4):14–23, oct 2009.

[126] Robert Sedgewick. *Algorithms in C, part 5: graph algorithms*. Pearson Education, 2001.

[127] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K Reiter, and Guangyu Shi. Design and Implementation of a Consolidated Middlebox Architecture. *NSDI'12 Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, page 24, 2012.

[128] Jaydip Sen. Visions of 5G Communications. *The ENVISION Journal of Tata Consultancy Services Ltd.*, 1(1):4 – 15, 2009.

[129] Muhammad Shahbaz, Sean Choi, Ben Pfaff, Changhoon Kim, Nick Feamster, Nick McKeown, and Jennifer Rexford. PISCES: A programmable, protocol-independent software switch. *SIGCOMM 2016 - Proceedings of the 2016 ACM Conference on Special Interest Group on Data Communication*, pages 525–538, 2016.

[130] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: Network processing as a cloud service. In *SIGCOMM'12 - Proceedings of the ACM SIGCOMM 2012 Conference Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2012.

[131] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[132] Thomas Soenen, Wouter Tavernier, Didier Colle, and Mario Pickavet. Optimising microservice-based reliable NFV management & orchestration architectures. *2017 9th International Workshop on Resilient Networks Design and Modeling (RNDM)*, pages 1–7, 2017.

[133] Hui Suo, Jiafu Wan, Lian Huang, and Caifeng Zou. Issues and challenges of wireless sensor networks localization in emerging applications. *Proceedings - 2012 International Conference on Computer Science and Electronics Engineering, ICCSEE 2012*, 3:447–451, 2012.

[134] Tarik Taleb, Konstantinos Samdanis, Badr Mada, Hannu Flinck, Sunny Dutta, and Dario Sabella. On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration. *IEEE Communications Surveys and Tutorials*, 19(3):1657–1681, 2017.

[135] D. L. Tennenhouse and D. J. Wetherall. Towards an active network architecture. In *Proceedings - DARPA Active Networks Conference and Exposition, DANCE 2002*, 2002.

[136] D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall, and G.J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, jan 1997.

[137] The Zeek Project. The Zeek Network Security Monitor, 2020.

[138] Reza Tourani, Austin Bos, Satyajayant Misra, and Flavio Esposito. Towards security-as-a-service in multi-access edge. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing - SEC '19*, pages 358–363, New York, New York, USA, 2019. ACM Press.

[139] Fung Po Tso, Konstantinos Oikonomou, Eleni Kavvadia, and Dimitrios P. Pezaros. Scalable Traffic-Aware Virtual Machine Management for Cloud Data Centers. In *2014 IEEE 34th International Conference on Distributed Computing Systems*, pages 238–247. IEEE, jun 2014.

[140] Fung Po Tso, David R White, Simon Jouet, Jeremy Singer, and Dimitrios P Pezaros. The glasgow raspberry pi cloud: A scale model for cloud computing infrastructures. In *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops*, pages 108–112. IEEE, 2013.

[141] J. Van Der Merwe, A. Cepleanu, K. D'Souza, B. Freeman, A. Greenberg, D. Knight, R. Mcmillan, D. Moloney, J. Mulligan, H. Nguyen, M. Nguyen, A. Ramarajan, S. Saad, M. Satterlee, T. Spencer, D. Toll, and S. Zelingher. Dynamic connectivity management with an intelligent route service control point. *Proceedings of the 2006 SIGCOMM Workshop on Internet Network Management, INM'06*, 2006:29–34, 2006.

[142] Luis M. Vaquero and Luis Rodero-Merino. Finding your Way in the Fog. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, oct 2014.

[143] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems*, pages 1–17, 2015.

[144] Petra Vizarreta, Massimo Condoluci, Carmen Mas Machuca, Toktam Mahmoodi, and Wolfgang Kellerer. QoS-driven function placement reducing expenditures in NFV deployments. *IEEE International Conference on Communications*, 2017.

[145] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser. Active messages. *ACM SIGARCH Computer Architecture News*, 20(2):256–266, may 1992.

[146] Junjue Wang, Ziqiang Feng, Zhuo Chen, Shilpa George, Mihir Bala, Padmanabhan Pillai, Shao Wen Yang, and Mahadev Satyanarayanan. Bandwidth-efficient live video analytics for drones via edge computing. *Proceedings - 2018 3rd ACM/IEEE Symposium on Edge Computing, SEC 2018*, pages 159–173, 2018.

[147] Zhenji Wang, Dan Tao, and Zhaowen Lin. Dynamic Virtualization Security Service Construction Strategy for Software Defined Networks. *2016 12th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN)*, (November 2010):139–144, 2016.

[148] Bing Wu, Jianmin Chen, Jie Wu, and Mihaela Cardei. A Survey of Attacks and Countermeasures in Mobile Ad Hoc Networks. In *Wireless Network Security*, pages 103–135. Springer US, Boston, MA, 2007.

[149] Hongjing Wu, Yan Zhang, Huiran Yang, Guangxi Yu, and Jiuyue Cao. Virtualized Security Function Placement for Security Service Chaining in Cloud. In *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, volume 2018-Decem, pages 628–637. IEEE, dec 2019.

[150] Ming Xia and Howard Green. SOLuTIoN : SDN-based OpticaL TraffIc steering for NFV. *HotSDN 2014*, pages 227–228, 2014.

[151] William Xu. GIV 2025 Unfolding the Industry Blueprint of an Intelligent World. *Huawei Tech.*, 2018.

[152] L. Yang, R. Dantu, T. Anderson, and R. Gopal. Forwarding and Control Element Separation (ForCES) Framework. Technical report, IETF Network Working Group, 2004.

[153] Jin Y Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.

[154] Shanhe Yi, Cheng Li, and Qun Li. A Survey of Fog Computing: Concepts, Applications and Issues. In *Proceedings of the 2015 Workshop on Mobile Big Data - Mobidata '15*, 2015.

[155] Ying Zhang, Neda Beheshti, Ludovic Beliveau, Geoffrey Lefebvre, Ravi Manghirmalani, Ramesh Mishra, Ritun Patneyt, Meral Shirazipour, Ramesh Subrahmaniam, Catherine Truchan, and Mallik Tatipamula. StEERING: A software-defined networking for inline service chaining. In *2013 21st IEEE International Conference on Network Protocols (ICNP)*, pages 1–10. IEEE, oct 2013.

[156] Masahiro Yoshida, Wenyu Shen, Taichi Kawabata, Kenji Minato, and Wataru Imajuku. Morsa: A multi-objective resource scheduling algorithm for nfv infrastructure. In *The 16th asia-pacific network operations and management symposium*, pages 1–6. IEEE, 2014.

[157] Tianlong Yu, Seyed K Fayaz, Michael Collins, and Vyas Sekar. PSI: Precise Security Instrumentation for Enterprise Networks. *Proceedings of NDSS 2017*, 2017.

[158] Tianlong Yu, Vyas Sekar, Srinivasan Seshan, Yuvraj Agarwal, and Chenren Xu. Handling a trillion (unfixable) flaws on a billion devices. *Proceedings of the 14th ACM Workshop on Hot Topics in Networks - HotNets-XIV*, pages 1–7, 2015.

[159] Bruno Bogaz Zarpelão, Rodrigo Sanches Miani, Cláudio Toshio Kawakani, and Sean Carlisto de Alvarenga. A survey of intrusion detection in Internet of Things. *Journal of Network and Computer Applications*, 84(February):25–37, 2017.