On the Placement of Security-related Virtualised Network Functions over Data Center Networks

Abeer Farouk Tawfeek Ali

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

August 2020

Abstract

Middleboxes are typically hardware-accelerated appliances such as firewalls, proxies, WAN optimizers, and NATs that play an important role in service provisioning over today's data centers. Reports show that the number of middleboxes is on par with the number of routers, and consequently represent a significant commitment from an operator's capital and operational expenditure budgets. Over the past few years, software middleboxes known as Virtual Network Functions (VNFs) are replacing the hardware appliances to reduce cost, improve the flexibility of deployment, and allow for extending network functionality in short timescales.

This dissertation aims at identifying the unique characteristics of security modules implementation as VNFs in virtualised environments. We focus on the placement of the security VNFs to minimise resource usage without violating the security imposed constraints as a challenge faced by operators today who want to increase the usable capacity of their infrastructures. The work presented here, focuses on the multi-tenant environment where customised security services are provided to tenants. The services are implemented as a software module deployed as a VNF collocated with network switches to reduce overhead. Furthermore, the thesis presents a formalisation for the resource-aware placement of security VNFs and provides a constraint programming solution along with examining heuristic, meta-heuristic and near-optimal/subset-sum solutions to solve larger size problems in reduced time.

The results of this work identify the unique and vital constraints of the placement of security functions. They demonstrate that the granularity of the traffic required by the security functions imposes traffic constraints that increase the resource overhead of the deployment. The work identifies the north-south traffic in data centers as the traffic designed for processing for security functions rather than east-west traffic. It asserts that the non-sharing strategy of security modules will reduce the complexity in case of the multi-tenant environment. Furthermore, the work adopts on-path deployment of security VNF traffic strategy, which is shown to reduce resources overhead compared to previous approaches.

Acknowledgements

With a lot of love and appreciation, I would like to express my grateful appreciation to the people who help me accomplish this work. First, I would like to thank my supervisor Dimitrios P. Pezaros for his continues motivation, guidance and support. Thanks also to my second supervisor Christos Anagnostopoulos for your knowledge and discussions. I also want to thank the UK Engineering and Physical Sciences Research Council (EPSRC) for funding this work. To all my friends in Egypt and Glasgow, Thanks for your friendship and encouragement, this work will never be completed without you. Finally thanks to my Mam, sisters and brother for your love and support. For my Dad, you are not among us, but you are always with me.

Table of Contents

List of Tables

List Of Abbreviations

AIDS Anomaly-base Intrusion Detection Systems.

BFD Best Fit Decreasing.

DC Data Center.

DDoS Distributed Denial of Service.

DoS Denial of Service.

DPI Deep Packet Inspection.

ECMP Equal-Cost Multi-Path.

FFD First Fit Decreasing.

ICT Information and Communications technology.

IDS Intrusion Detection System.

IPS Intrusion Prevention System.

LP Linear Programming.

MIDS Misuse-base Intrusion Detection Systems.

NF Network Function.

NFV Network Function Virtualisation.

PPS Packet Per Second.

SDN Software-Defined Networking.

VMP Virtual Machine Placement.

VNE Virtual Networks Embedding.

VNF Virtualised Network Function.

VSBPP Variable Sized Bin Packing Problem.

List of Figures

Chapter 1

Introduction

1.1 Overview

Large-scale Data Centers (DCs) are the underlying infrastructures that provide virtualised compute, network, and storage resources in an elastic manner. Therefore, many organisations have outsourced their information communications technology (ICT) provisioning to the cloud and successfully reduce their capital and operational expenditure. However, the increase in the size of data centers causes a phenomenal increase in operational cost for services providers. This combined with the operator's profit-awareness, motivate the research in different aspects of managing data centers such as resource management, energy efficiency, networking, and security.

Network services in data centers such as firewalls, caches, proxies, intrusion detection systems, WAN accelerators, etc., have been deployed as high-speed vendor-specific hardware-based middleboxes physically hardwired to the network infrastructure of data centers. While surveys show the number of middleboxes is equal to or exceeding the number of routers at all network sizes, estimations of the initial hardware cost of middleboxes for an enterprise network (between 10,000 and 100,000 hosts) reached $1m every 5 years. Thus, much of the research conducting on managing data centers are directed to middleboxes, security in particular, where numbers show that security middleboxes can reach up to more than 30% of

the network middleboxes [1].

As ICT is moving to the cloud, modern data centers underpin the *-as-a-Service paradigm with increasing (in-the-cloud) services offered to users by service providers. However, hardware middleboxes limit cloud service provider ability to offer network functions as services where they are suffering from expensiveness, deployment inflexibility, inefficient resources management, vendor-specific and limited functionality [2–4]. Furthermore, it has limited ability to provide customised services. On the other hand, softwarizing middleboxes have been offered as a solution to the hardware-based middleboxes problems.

Through exploiting the newly emerged Software Defined Networking (SDN) and Network Function Virtualisation (NFV), service providers can efficiently manage the softwarised network functions in the cloud paradigm. Hence, VNF enables data center operators to manage and orchestrate virtualised services/software middleboxes as Virtualised Network Functions (VNFs), providing the deployment flexibility and the efficient provisioning of resources that hardware middleboxes lack. While SDN will provide the global view and centralised control of the network to orchestrate the VNFs and introduce programmability as well. With these two complementary technologies, virtualised services will reduce the operational expenditure and improve the utilisation of existing resources.

Users of a multi-tenant environment run different applications which require different levels of security per application. For example, a web server may require protection against HTTP flooding attacks, while critical servers may require deep packet inspection and/or a combination of signature-based and anomaly-based intrusion detection. Over the cloud, security services are offered by cloud Services Providers (CSP) or third-party companies to satisfy the need for customised security services. As the use of virtualised middleboxes becomes more widespread, research is focusing on the different aspects of managing network functions in virtualised environments. Nevertheless, only a few consider the distinct requirements and constraints related to security functions in a multi-tenant environment.

This dissertation investigates how the newly emerging technologies can leverage virtualised data centers to address the placement of security modules as virtualised network functions in

a multi-tenant environment with the objective of increasing usable capacity. Furthermore, it presents an analysis of security modules processing of traffic that leads to a classification of security modules based on the required granularity of traffic processing. The classification introduces new constraints that impose a resource overhead to the management of security functions. While other work addressing the efficient management of network functions aims at utilising resources or increasing performance measures, these approaches have limitations when it comes to deploying security functions, and the work presented in this thesis demonstrates these limitations.

Furthermore, this thesis advocates on-path deployment to reduce resource consumption and a non-sharing strategy to reduce complexity. Combined with VNF and SDN, this work allows service operators to provide resource-efficient on-demand customised security services for tenants by deploying security modules as VNFs in the multi-tenant environment. Moreover, this work provides the mathematical analysis to the placement problem of security services and time-optimised solutions targeting at saving the infrastructure's computing and communication resources.

To conclude, the deployment of network functions as VNFs introduces flexibility and dynamism as uprising demand in modern multi-tenant environments such as the cloud. This work asserts there are unconsidered and vital constraints to the deployment process of security network functions compared to other types of functionality. These constraints will restrict the placement operation and increase resource consumption of the deployment. The work presented here identifies these security constraints and proposes a placement framework that satisfies these constraints while maintaining efficient management of the resources by adopting on-path deployment and introducing a heuristic algorithm as a time-optimised solution to the problem.

1.2 Contributions

The contributions of this thesis are as follows:

- Identifying the unique constraints of security functions placement based on traffic analysis requirement and traffic directional, compared to other network functions.
- The design of a resource-aware placement strategy that satisfies the traffic constraints of the security functions different classes.
- The design of a resource-aware placement framework for customised security services in multi-tenant data centers where security modules are implemented as on-path VNFs.
- The formulation of one-dimensional and two-dimensional resources implementations of the security VNFs placement problem with the objective of increasing usable capacity.

1.3 Thesis Outline

The remainder of this dissertation is structured as follows:

- **Chapter 2** describes the legacy security solutions and their deployment by network providers, and the inherent limitation of these legacy tools in modern networks. It discourses the motivation for the creation and development of Virtualised Network Function and Software Defined Networking, and covers the trade-offs of softwarising network functions compared to legacy middleboxes. Then, it presents the motivation for developing orchestration systems for VNFs. The chapter details current platforms for orchestrating VNFs and latest research in the placement problem. Then, it focuses on security functions orchestration and discusses their current issues and limitations.
- **Chapter 3** presents the design of the proposed security placement framework. It describes the framework architecture and characteristics. It introduces the classification of the security modules based on traffic processing requirements and then the placement strategy adopted by the framework based on that classification. Then, it describes the two-dimensional resource-aware placement problem of the framework along with

its mathematical formulation. Furthermore, the chapter introduces heuristic, meta-heuristic, near-optimal methods as solution to the placement problem. Finally, it introduces a one-dimensional resources implementation to the problem.

- **Chapter 4** provides the technical aspects of the implementation of the design discussed in Chapter 3 on fat-trees. The chapter begins by describing the architecture of a fat-tree data center, traffic routing, and the implementation of the proposed framework of on-path VNFs. Then, it presents the constraint programming implementation of the placement problem by the CPlex optimiser. Then, it discusses the implementation of heuristic, meta-heuristic, near-optimal algorithms on fat-tree. Finally, a linear programming solution to the one-dimensional implementation of the placement problem is presented.

- **Chapter 5** presents a comprehensive evaluation of the resource-aware security VNF placement framework. Through simulation, the placement framework and solution methods are evaluated to determine the optimality of solutions compared to the constraint programming solution. Then, it extended the evaluation to different characteristics such as network size, number of modules, success rate and class type distribution. Moreover, it presents the optimality gap analysis and execution time of the solution methods. Finally, it evaluates the scalability of the framework with the number of requests.

- **Chapter 6** gives a summary of the contributions and findings of this work, and explores potential research directions and future work.

Chapter 2

Related Work

2.1 Overview

Data centers (DCs) important role in modern IT infrastructure can be contributed to 1) provide the computational power, storage, and applications necessary to support various enterprise business 2) numerous services run by a single infrastructure, in the contradictory to previous model where each service had its own server to be operated on 3) reduce capital and operational expenditure for businesses due to the economics of scale principle 4) ability to process billions of Internet transactions every day [5]. This led to the deployment of large data centers with thousands of servers by renowned ICT organisations such as Amazon, Microsoft, and Google to offer cloud computing services to a wide range of users and businesses. However, the increase in the size of data centers causes a phenomenal increase in operational cost, this combined with the operator's profit-awareness motivate the research in the management of data centers such as resource management, energy efficiency, networking, and security [6].

Security in data centers is accomplished by installing dedicated security components by system administrators, such as anti-malware, firewalls, Intrusion Detection and Prevention Systems (IDS/IPS) that usually perform Deep Packet Inspection (DPI) to detect attacks. These security components are commonly hardware-based middleboxes deployed in fixed loca-

tions across the network [7]. While many companies (e.g., Cisco, Juniper, Fortinet, Blue Coat, IBM, Radware, and Intel security) offer line speed appliances that provide firewall, IDS, IPS, and DPI functionality, these appliances are allocated manually based on a static risk management process [8].

These legacy hardware security systems suffer from many problems such as lack of deployment flexibility, limited functionality, high cost, and inefficient management of resources. Many of these problems are inherited from hardware-based systems. As ICT is increasingly outsourced to the cloud, middleboxes start the transition with the increasing number of virtualised network appliances such as WAN optimiser (e.g. virtual VX from SilverPeak [9] and SteelHead from Riverbed [10]), Firewalls (e.g. ASAv from Cisco [11] and XG from Sophos [12]), and IDPS (e.g. Snort [13] and Suricata [14]). There is also in-the-cloud network services offered by cloud Services Providers (CSP) or third party companies. The virtualised security services will provide the same protection that hardware-based systems provide combined by the high performance and efficiency of cloud services. Although with the increase in the size of data centers, the management of theses virtualised services together with data center infrastructure resources becoming more of a complex task that an inefficient one can cause performance degradation and/or reduce turnover [3]. However, recent technologies such as SDN and VNF can be exploited to accomplish efficient management of infrastructure resources combined with facilitating the process of managing the virtualised services.

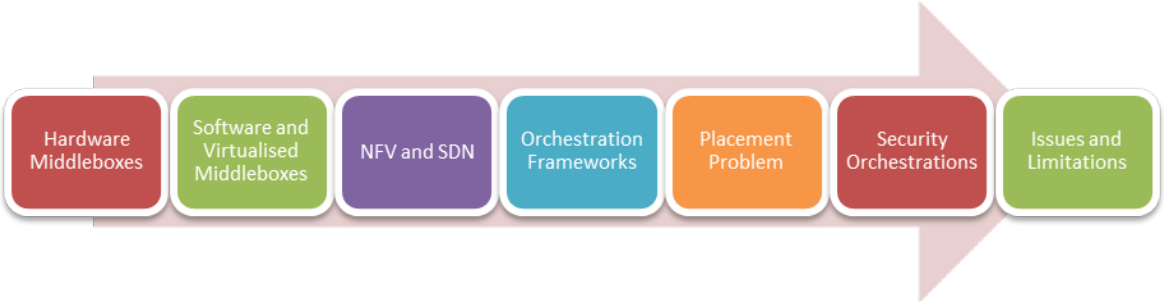


Figure 2.1: Literature Survey

We inspect the state-of-the-art work in the related fields to be able to comprehend the contribution context. With numerous work that has been done in the area of VNF orchestration, it

is necessary to examine this work and identify how it can be applied in the context of security functions. The organisation of the literature Survey is shown in Figure 2.1. First, it details hardware and software middleboxes and demonstrates their comparison. Then, it introduces the Network Functions Virtualisation and Software-Defined Networking as the tools exploit to deploy and manage software middleboxes as Virtualised Network Functions (VNFs) in virtualised data centers. Then, it discusses the frameworks addressing the management and orchestration of VNFs then placement strategies implemented. Then, we discuss the work related to security network functions such as the security functions orchestrations framework. Finally, it discussed the issues and limitations of orchestrating security functions of previous work of security VNFs security functions. Therefore, this chapter is organised as follows:

Section 2.2 defines security threats and their classification and examines DDoS attack as an example. Then, it illustrates security tools classification and examples and discusses their threat detection techniques. Finally, it describes the traffic preprocessing phase of security functions.

Section 2.3 details the best practices of hardware security tools deployment by network providers. Then, it demonstrates the limitations of these legacy tools in modern networks. Then, it presents software middleboxes and network function virtualisation and highlights their potentials for deploying security modules.

In Section 2.4, we discuss the problem of orchestrating virtual network functions, then examine the characteristics of SDN-enabled networks that support solving the problem. Then, we outline the frameworks of orchestration VNFs and their proposed implementation. Then, we discuss the latest research in VNF placement and the short-comes.

Section 2.5 discuss potentials that VNFs offer to security functions. Then, it illustrates the rise of customised security services in multi-tenant virtual environments. Then, it demonstrates frameworks of orchestrating security functions and placement strategies. Then, it details the current issues and limitations for orchestrating security functions in multi-tenant environments. Finally, we summarise the chapter in Section 2.6.

2.2 Network Security

As a real-time system, computers have some security threats that are commonly known as computer attacks and defined as any malicious act against one or more computer system. Bhuyan et al. in [15] classify computer attacks based on their nature as; Viruses, Trojans, Worms, Denial of Service (DoS), Network attacks, Physical attacks, Password attacks, Information gathering attacks, User to Root attacks (U2R), Remote to Local attacks (R2L) and probe attacks. The common purposes of attacks are usually accessing, gathering, manipulating data or driving the system into an inaccessible state for legitimate users [15]. Here, the term network attack will be used to refer to any act that exploits network communication in a malicious intent against a host, a subnet of hosts, or a network component in a computer network.

2.2.1 Distributed Denial of Services (DDoS)

For example, Denial of Services (DoS) attacks are one of the major network attacks that networks are facing today. They mainly aim at disturbing the normal behaviour of a system by over-consuming compute or network resources at the victim site, making it inaccessible or slow to legitimate users and in some severe cases causing entire system failures. The attacks usually are made by sending large volumes of traffic that leave the victim site in an unstable state, causing the system to deny some or all the services to legitimate users.

Distributed Denial of Service (DDoS) attacks are DoS attacks with multiple synchronised attacker sources that add more bandwidth and consequently amplify the damage [16]. To launch a powerful DDoS, attackers usually take control of a large number of machines (zombies or bots) by infecting them with malware which allows them to control the machine by sending instructions through a handler program such as Internet Relay Chat (IRC) and HTTP requests [17, 18]. Bots make it harder to detect attacks due to their distributed nature. Today, almost all attacks are distributed in nature which makes DDoS a major threat to any computer system connected to a network. DDoS attacks can be classified based on different

criteria such as attack layer, launching method, and vulnerability exploited. In [19], Specht et al. suggest a classification of DDoS attacks based on their impact on system resources which as follows:

1. **Bandwidth depletion attacks:** the victim's network is flooded with traffic preventing legitimate users from reaching the victim, e.g., flooding attacks (UDP and ICMP) and amplification attacks (Smurf and Fraggle).

2. **Resource depletion attacks:** attack traffic consumes the victim resources preventing it from processing legitimate user requests, such as protocol exploitation attacks (e.g., TCP SYN) and malformed packet attacks (e.g., Land attack).

3. **Application-level attacks:** server resources (e.g., sockets, memory, CPU cycles) are exhausted, or a vulnerability in the application layer protocol is exploited, such as HTTP fragmentation and HTTP GET attacks [18, 20].

Recently, there has been a substantial increase in DDoS attacks in volume and rate. In March 2018, **GitHub**[1] reported the largest DDoS attack in history till now. A reported 1.35 Tbps DDoS attack that makes the famous hosting website unavailable for around 10 minutes. The volumetric attack was created by an unusual attack method that exploits a bug in the most-widely used Memcached database servers to create an amplified traffic attack with a response that can be 51,200 times bigger than the original request [21]. Moreover, the largest Packet Per Second (PPS) DDos attack was recorded in April 2019 by Imperva[2] which peaked at 580 PPS while GitHub peaked at 129.6 MPPS. DDoS attacks can also be multi-vector, in November 2016, Akamai[3] the Content Delivery Network (CDN) and Security provider confirmed a 5-day attack on a website that peaked at 623 Gbps generated traffic, consisting of six DDoS attack vectors: GRE floods, SYN floods, NTP amplification, and ACK floods at the network level, and both PUSH and GET floods at the application layer. Furthermore,

[1] https://github.com/
[2] https://www.imperva.com
[3] https://www.akamai.com/

Neustar[4] reported in their Cyberthreats and Trends Report an 180% increase in DDoS attacks in 2019 compared to 2018. This increase can be contributed to two main factors:

1. Indirect attacks are massive volumes of aggregate traffic generated by small initial attack vectors which make it easy to generate and hard to detect [22]. For example, the most famous DNS amplification attack exploits open DNS resolvers to issue requests with the victim's spoofed IP address. In the attack on Spamhaus in 2013, a 36-byte DNS malicious request converted to more than 3,000-byte response as shown in Figure 2.2, and results in aggregate 75 Gbps attack volume that launched with 30,000 unique DNS resolver involved [23].

2. The outbreak of botnet services (DDoS-as-a-service or malware-as-a-service) that become more powerful and inexpensive as described below.

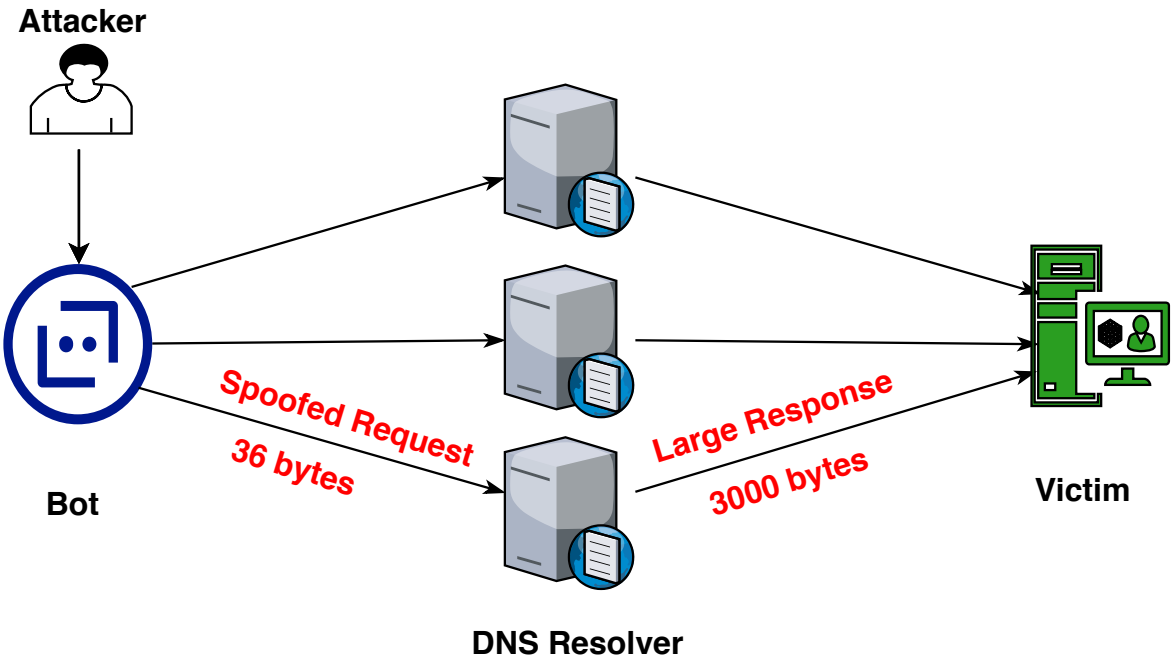


Figure 2.2: Amplification Attack

The recent outbreak of botnets provides attackers with a powerful launching platform for their attacks which can be attributed to **1)** Services like DDoS-as-a-service and malware-as-a-service that enable even inexperienced attackers to create a powerful attack vector with

[4]https://www.home.neustar/

little expense [22, 23]. **2)** cloud clones of VM instances allow an attacker with usually hijacked cloud account to easily and rapidly creates bots by duplicating instances that do not need much memory or disk space [22]. **3)** The wide penetration of insecure consumer devices (e.g., tablets, Smartphones, laptops, and IoT devices) with broadband connectivity capabilities [22]. For example, 150K compromised IoT devices were used to launch a 623 Gbps attacks on the *Kerbs on security*[5] website and Dyn[6] The Internet infrastructure provider company. The attack on Dyn disrupted services such as Netflix, Twitter, Amazon, Spotify, Reddit, CNN, PayPal, Pinterest and Fox over the east coast of the United States. The attacks use the Mirai malware which took control of IoT devices with a weak/default password. This kind of malware demonstrates the ability to launch attacks with billions of devices around the world. Reports claim that the Dyn attack reached 1 Tbps in volume.

Furthermore, The rapid increase of volume and rate of attacks transforms the cloud from a promising solution to mitigate the effects of DDoS attacks due to the over-provisioning of resources to a potential target. With attack traffic reaching 1Tbps, even global cloud service providers are being tested when successful attacks can take down parts of the Internet as seen on the Dyn attack. Furthermore, powerful bots can cause a multiplied damage that people making bots went so far to hire developers with a unique brand- and vendor-specific expertise to create bots that can avoid anti-bot measures and escape detection which reported by Akamai in Internet state security report in 2019 [24].

Since many corporate and global ICT systems are moving their daily operations to the cloud such as banking transactions, government services, online shopping, entertainment... etc. to reduce their capital and operational expenditure [22], cloud services are increasingly becoming targets of attacks. While services offered by cloud service providers (CSP) are offered in a scalable, elastic and always-on manner, they are extremely prone to security vulnerabilities which cause downtime, economic loss, and reputation damage to the infrastructure, service and application providers.

[5] https://krebsonsecurity.com/
[6] https://dyn.com/

2.2.2 Classification

In enterprise systems, the primary defence system against network threats is a combination of prevention, detection and mitigation techniques. Prevention is usually done by filtering any suspicions traffic before it reaches the destination hosts. For instance, prevention can be done using Turing tests in the form of CAPTCHAs or puzzles to identify legitimate users and block spoofed traffic [25]. Detecting attacks is accomplished by installing dedicated security components by system administrators, such as anti-malware, firewalls, Intrusion Detection or Prevention Systems (IDS/IPS) that usually perform Deep Packet Inspection (DPI). These security components are commonly hardware-based middleboxes deployed in fixed locations across the network [7]. On the other hand, mitigation techniques are based on filtering attacker's traffic and do not guarantee full elimination of the attack [18]. As a common mitigation technique, malicious traffic identified by the detection process can be filtered in upstream routers to mitigate the attack; however, this process is prone to false positives and results in legitimate traffic filtered as a malicious one. Alternative mitigation techniques aim at surviving attacks by scaling up resources until the attack is over. However, this can only be used on infrastructures where scaling is provided on-demand (e.g., in clouds).

Furthermore, some of the solutions mentioned above are not effective against massive DDoS attacks that can scale up to overwhelm most traditional on-premises equipment and resources available at cloud providers as seen in recent attacks such as Dyn attack. As a popular option, third-party mitigation services (e.g., Cloudflare, Akamai) can be used to mitigate such attacks, as they have massive amounts of network bandwidth and DDoS mitigation capacity at multiple locations around the world that can absorb and filter any amount of network attacks. Using these services is effective since these providers are fine-tuned to cope with extremely high demand but are often expensive due to their infrastructural requirements. They can also raise privacy concerns since user traffic is redirected to third-party servers [26].

As the ineffectiveness of the legacy on-premise detection systems is caused by the inherited problems of hardware-based middleboxes, resolving some of these problems such as the inflexible deployment can increase the system's ability to handle massive volume attacks

without using expensive mitigation add-on services. Besides, most security systems in the market do not belong to only one of the previous classes but usually combine more than one. In the following, we discuss firewall and intrusion detection as the two most widely deployed security solutions currently in enterprise networks to process network traffic.

2.2.3 Firewalls

A firewall function is used to deny or allow specific traffic based on IP addresses, protocols or ports. It is installed at the entry points of the system to examine all egress and ingress traffic [27]. Firewalls can be categorised as: Access Control List (ACL)-based stateless firewalls that evaluate packet contents statically; stateful firewalls that keep track of the bidirectional state of network connections (e.g., TCP streams, UDP communication) [28], travelling across in both directions and only those forming a proper connection are permitted to pass through the firewall; Proxy firewalls that analyse the protocol syntax by breaking up client/server connection [27]. Some firewalls apply packet filtering as mitigation or prevention system.

2.2.4 Intrusion Detection System (IDS)

An intrusion detection system (IDS) tries to detect intrusion or threats through monitoring and analysing events that occur in a computer system or a network. Intrusion Prevention Systems (IPS) is an active IDS that also detect intrusion but can also react to stop them from damaging the system, for example, stop the attack by dropping the packets, reconfiguring the firewall or changing the attack' content [29]. A typical IDS is organised in modules including data collection, preprocessing, detection and reporting (alarm) modules, and in the case of an IPS, a countermeasure (response) module is included [30]. The detection module is responsible for providing as much information as possible regarding the intrusion detected to support forming a response to stop or mitigate the intrusion or to be used later for further analysis to discover the vulnerabilities of the system. IDS provide information such as source(s) and/or destination(s) of the attack, type of attack, attack volume and period.

An IDS can be classified according to multiple characteristics such as (Detection method, Deployment location, structure, time of detection ..etc.) [30–37], as shown in Figure 2.3

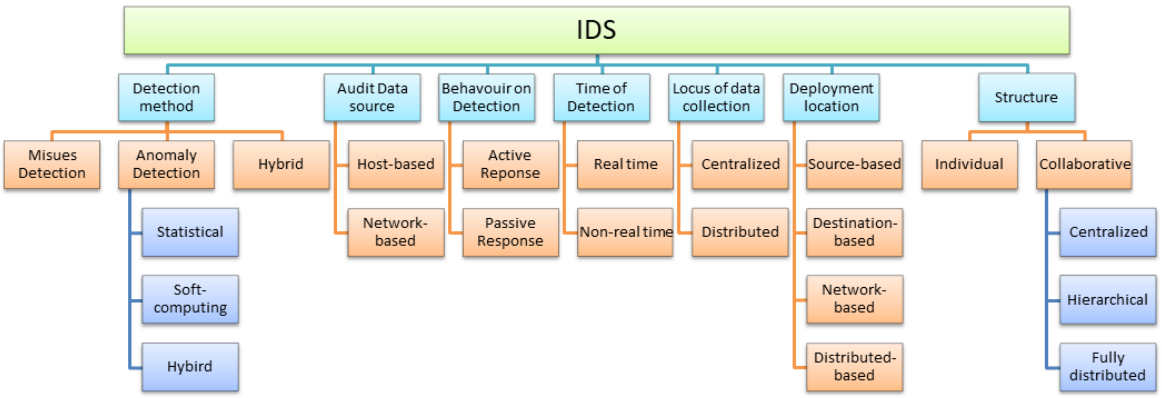


Figure 2.3: IDS Classification.source[34]

The detection algorithm is the core of the IDS that defines how the intrusion will be detected and subsequently determines other characteristics of the IDS such as which data to be collected and in what form, time of detection where some detection algorithms are time-consuming to be running in real-time mode. There are three main categories of IDS: misuse-based, anomaly-based and hybrid approaches.

2.2.4.1 Misuse-based Intrusion Detection Systems (MIDS)

Misuse-based Intrusion Detection Systems (MIDS) compare the stored pattern of known attacks to the analysed data and report intrusions if a match is found. It has a low false positive rate, but it is clear that it cannot detect new intrusions (zero-day attacks) or attacks with no corresponding patterns in its knowledge base [36]. Furthermore, defining the attack signatures is hard work since it is challenging to write a signature to detect all variance of an attack [38]. In [39], data mining techniques are used to generate the signatures; however, it can be time-consuming. Most MIDSs are network-based, using features derived from packet headers, payload, or both, For example, SNORT [13] used signatures derived from header data (source address, destination address and ports) and optional content data (payload, meta-

data). However, host-based features like system calls can also be modelled to represent an attack pattern [38]. A MIDS has a huge knowledge-base due to the enormous number of attacks discovered every day and their variants. Therefore, MIDS are computationally expensive with respect to time and requirements to match all signatures to the sampled data, especially if payload features are used causing performance to vary depending on the pattern matching technique implemented. This companionability can result in losing packets in high-speed links because the system can not process incoming data at line speed which makes it impractical for high-speed networks [33, 37]. Furthermore, a knowledge-base with the latest attack patterns must be maintained and updated periodically or whenever a new attack has been identified. The most popular and widely used network MIDS are Snort and Bro [13, 40].

2.2.4.2 Anomaly-based Intrusion Detection Systems (AIDS)

Anomaly-based Intrusion Detection Systems (AIDS) model the normal behaviour of the system and report an intrusion if the observed behaviour deviates from the normal model. MIDS and AIDS can be distinguished through the difference between the two words attacks and anomaly: while an attack is an action that can be defined explicitly by a signature or a rule to be matched in MIDS, the anomaly is a deviation from what normal behaviour is expected to be [36]. As a result, anomaly-based IDS can detect zero-day attacks or variants of known attacks if their behaviour represents a deviation from the modelled behaviour while a MIDS can only detect attacks represented in their attack databases. Nevertheless, AIDS has a high false positive rate due to those legitimate events that can cause deviation from the stored normal model such as, the stored model is inaccurate or out of date and in various cases some events may seem abnormal although it is a legitimate event like a flash crowd. The anomaly detection is performed in two phases: the training phase and the detection phase [20]. The training phase is used to build a model for normal behaviour, and the detection phase is the process of comparing the stored model to the observed model and generates an alarm if any deviation is detected. Stateful protocol analysis detection methods

build a profile of protocols normal activity instead of profiling network or host behaviour. It can track the state of the network, transport and application protocol by building profiles from vendor-developed protocol implementations [29]. Methods used to build the normal behaviour model and compare it to new events can be classified as statistical, soft-computing techniques, and hybrid methods [20].

In statistical techniques, a dataset is used to construct a profile for the system monitored activity; the profile consists of intensity or distributed measures and is stored as the normal profile of the system. The system calculates the current profile and compares it to the stored profile. If the changes in measures exceed a certain threshold, an alarm is raised. The system updates the stored profile periodically to reflect system changes. Statistical techniques can detect unknown attacks; however, setting threshold values for the profile measures is a complicated process, and also attackers can train the system through slow attacks to avoid detection [41]. One of the earliest implementations of statistical-based techniques is Haystack [42], where profiles for users and groups were generated based on Gaussian random variables. Statistical techniques use a wide range of measures and approaches to build normal profiles and measure deviation such as mean, standard deviation, T2 test [43], entropy [17, 44], covariance matrix [45], outlier algorithms [46], Bayesian networks, etc. Soft-computing techniques are used to improve the accuracy of anomaly-based detection methods [27]. For example, Neural networks have been used in the case of incomplete datasets. In [47] a multi-layer neural network is built as a multi-class problem to identify the type of attack, not just the presence of one. Authors also propose a design to the optimal neural network with regards to the number of hidden layers. A Support Vector Machine (SVM) is used when a limited sample data is present. In [48], SVM outperforms neural network approach in the case of limited sampled data. In fuzzy logic techniques, fuzzy logic variables are used to represent system features [49]. Genetic algorithms have been used to select the best features to be used by other techniques. In [50], genetic algorithms are used to find the best-fit feature for a fuzzy algorithm. In [51], a Markov model approach is used to implement a host-based intrusion detection to model system calls.

2.2.4.3 Traffic Preprocessing

Traffic Preprocessing in a security tool is the process of constructing the features required by the detection algorithm. For example, in a stateful firewall, preprocessing involves reconstructing traffic connections and extracting parameters for comparing against the firewall rule set. In a network-based AIDS, preprocessing is more complex where network traffic can be represented by infinite features. These features can be classified as volumetric based or informative measures (flow variations based or probability measures), time-based or connection-based, packet-based or flow-based, header-based or content-based, single-connection or multiple-connection and more sophisticated measures can be used like entropy [35]. It is reported that this process can take up to 50% of the overall time of the detection system while the detection process can take up to 20% [37]. Thus, it has a significant impact on the security system performance.

Yet, the features selected for the detection phase determines the type(s) of detected attacks and the accuracy of detection. For example, most anomaly detection systems are flow-based (only use information derived from packet headers). They use flow and/or flow aggregate records as input to model the normal behaviour of a system. Such data only represents the interaction between nodes and does not carry all the traffic information, in particular, the payload information. However, flow data provide enough information to detect most attacks by examining communication patterns, periodic changes and temporal trends [33, 35]. However, some attacks cannot be detected unless the payload is examined. Payload attacks are more difficult and expensive to detect. It usually involves malicious content in the payload of IP/TCP packet, for example (e.g.server-side content attacks such as SQL-injection and cross-site scripting) [37]. On the other hand, some anomaly detector is built based on content-based features. For example, PAYL [52] is content-based AIDS built to detect zero-day worms. It is based on packet payload having an unusual byte frequency in the presence of a worm. It builds a model of frequency, variance and standard deviation of each the 256 of the byte possibilities for each destination port and length of the flow.

Another example, application-level DDoS attacks (e.g., HTTP flooding) are nearly invisible

from the flow level. Attack traffic is presented as legitimate TCP connections but drains the victim internal resources which can not be seen from the network view. Some of them can be detected using outbound instead of inbound traffic. The outbound bandwidth of the victim is saturated by responses to what seem to be legitimate requests causing volumetric changes in the outbound traffic that can be detected at the flow level [53]. However, some AIDS were proposed to detect application-layer attacks based on analysing the server request in the application-layer header. For example, web servers' DDoS attacks can be detected by observing features like request rate, download rate, uptime, downtime, page access rate, etc. [53]. In [54], an analysis of SQL statements between databases and the web application is used to detect SQL injection attacks. Thus, an application-layer DDoS detection algorithm uses features excluded from the application-layer header. As features selected to model the system define the coverage of the IDS (which attacks will be detected) and some attacks cannot be detected using certain features. Thus, some approaches try to find some common features that can be used to detect different types of attacks and reduce preprocessing. For example, scans cannot be detected using volumetric based methods, while most flooding DDoS attacks can [33]. Worms like Sapphire/Slammer cannot be detected on the flow level and require a payload analysis. In [33] an approach proposed observing the incoming/outgoing connection ratio of hosts can detect both DDoS and Scans and also detect worms in the scanning phase before launching the attack. Therefore, the accuracy of AIDS is highly dependent on the features selected to build the normal behaviour model [37].

Security functions characteristics such as features used in the detection process must be considered in the function deployment process to ensure accurate detection. However, the process of deploying legacy security functions by security experts is an ad-hoc process as detailed below, which makes considering this characteristic in the process a major challenge.

2.3 Security Systems

2.3.1 Hardware Middleboxes

Traditionally, security functions and other network functions such as (WAN optimiser, Caches and Proxies) have been implemented as hardware middleboxes. While many companies (e.g., Cisco, Juniper, Fortinet, Blue Coat, IBM, Radware, and Intel security) offer line speed appliances that provide firewall, IDS, IPS, and DPI functionality, these appliances are allocated manually based on a static risk management process where it deployed across different parts of the network to process the bulk of the ingress and egress traffic [8, 55, 56]. For example, a firewall function is installed at the entry points of the system to examine all egress and ingress traffic [27, 29]. In the case of IDS appliances, Cisco, for instance, recommends installing them in centralised positions around the protected network (e.g., between the network and the Internet to protect a connection with a business partner or to protect a specific Internet connection (e.g., a web server) [57]. A typical Enterprise network is shown in Figure 2.4 where a firewall is installed at the point connecting the system network to the Internet, followed by IDS to process malicious traffic. However, this approach prevents middleboxes from being efficiently managed and updated, as any maintenance on the network function requires all the traffic to be redirected to an alternative path until maintenance is completed. This approach is even more problematic in legacy infrastructure where the management protocols is limited, and most traffic redirection requires physically changing cabling of the network devices. Furthermore, the effectiveness of these approaches can be significantly limited in modern networks for example by the capabilities of the hardware or the fixed allocation of the security functions that reduce the system's ability to respond to attacks such as DDoS. We detail these challenges below:

2.3.1.1 Lack of Deployment Flexibility

The functionality of a defence system is measured by the accuracy of detection and performance stability over time. An efficient defence system must adapt to traffic changes (e.g.,

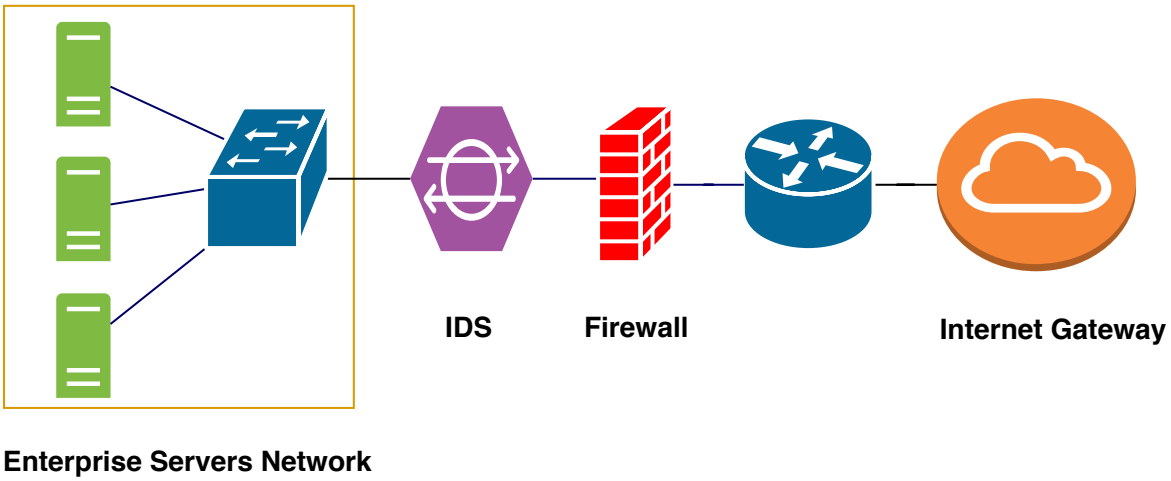


Figure 2.4: A Typical Enterprise Network

volume and distribution), infrastructure changes (e.g., failures, reconfiguration), and policy changes without degradation. These changes can occur under normal conditions or as a result of an attack. For a virtualised environment like the cloud, the rapid resource re-allocation such as VM migration is a typical change that a security system must adapt to and the adaptation must come into effect in short timescales [58]. The manual and ad-hoc placement of physical security appliances results in reconfiguration and maintenance of the network becoming a challenging process and affects the ability of the system to react rapidly to changes or respond to attacks [59]. Furthermore, as system administrators deploy middleboxes in specific locations, steering traffic to non-shortest paths can seriously affect the performance of the system [60].

2.3.1.2 Cost and Inefficient Management of Resources

To mitigate the problems mentioned above of deployment inflexibility and to increase system-wide fault tolerance, administrators tend to deploy more security middleboxes on network links which causes under-utilised, expensive middleboxes to be deployed across the network. The survey by Sherry et al. [2] shows that for an enterprise network (between 10,000 and 100,000 hosts), the hardware cost of middleboxes alone can reach $1m every 5 years. Figure 2.5 shows the number of middleboxes in Very large(>100k hosts), Large(10k-100k hosts), Medium(1k-10k hosts) and Small(<1k hosts) networks reported in survey conducted

in UC Berkeley [1]. It shows the numbers in a logarithmic scale of all middleboxes, router, Firewalls and IDSs in different network sizes. Moreover, it shows that the number of middleboxes is equal to or exceeding the number of Routers at all network sizes, while all security middleboxes represented in Firewalls and IDS can reach up to more than 30% of the network middleboxes. Furthermore, all very large networks in the survey had spent over $1m dollars on middleboxes hardware, while $50,000 was the spending of the top third of the small networks as reported by the survey. Moreover, because hardware middleboxes are not scaled up or down easily, the traditional approach is to provision for peak-demand in order to handle traffic spikes [61]. Thus, most middleboxes' resources are idle most of the time which increases the capital expenditure for under-utilised resources.

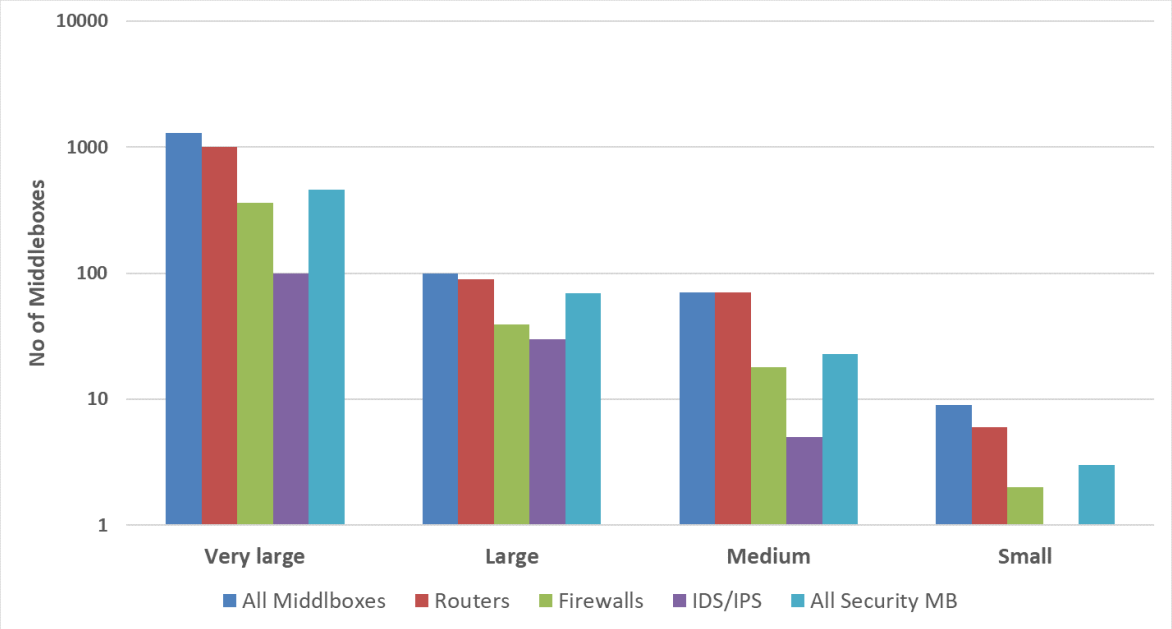


Figure 2.5: Number of Middleboxes in Enterprise Networks

2.3.1.3 Vendor Lock-in

The variations across vendor-specific middleboxes result in complex, specialised functions and different configuration interfaces for each vendor and device. Thus, security administrators are required to have per-vendor expertise for each type to effectively allocate and manage them, increasing expenses as a team of specialists is required to manage the appliances [62]. Besides, compatibility issues can arise in case of security system upgrade [2].

2.3.1.4 Limited Functionality

A security system must continuously adapt to respond to the latest threats [2] which includes changing the implemented security functionality such as, updating the attacks' signature-database, changing or extending the functionality of the security service itself. However, extending or updating a hardware-based appliance is usually very limited as there is a tight coupling between hardware capabilities (e.g., memory, TCAM, ASIC or NPUs) and the software running on them. Although reprogrammability for network equipment has been suggested by academic projects such as P4 [63], these projects have not reached widespread adoption among vendors.

2.3.2 Software Middleboxes

To mitigate the problems of legacy hardware-based middleboxes such as, expensiveness, vendor lock-in, deployment inflexibility, and lack of resource scalability [64], virtualised/-softwarised middleboxes have emerged such as WAN optimizers [9, 10], Firewalls [11, 12, 65, 66] and IDPS systems [13, 14, 67]).

Most modern networks support virtualisation. It allows heterogeneous architectures and applications to run on the same hardware as shown in Figure 2.6. Therefore, it has been used to optimise the usage of physical resources and to reduce expenses. One of their main features is on-demand resource allocation where virtual nodes can be dynamically instantiated and removed to satisfy changes in demand [6, 68]. Cloud computing is a paradigm that uses virtualisation to provide computing and networking resources as services. Usually, it uses multiple data centers in several geographic locations as the backbone of the system. As we mainly consider design aspect of the multi-tenant data center we will use the term data center to reference **virtualised multi-tenant data center** which is a virtualised data center architecture that is suitable for service deployment in a public or private cloud model [69].

In multi-tenant virtualised environments (e.g. cloud data center), Softwarised middleboxes are implemented as Virtualised Network Function (VNF). Network Function Virtualization

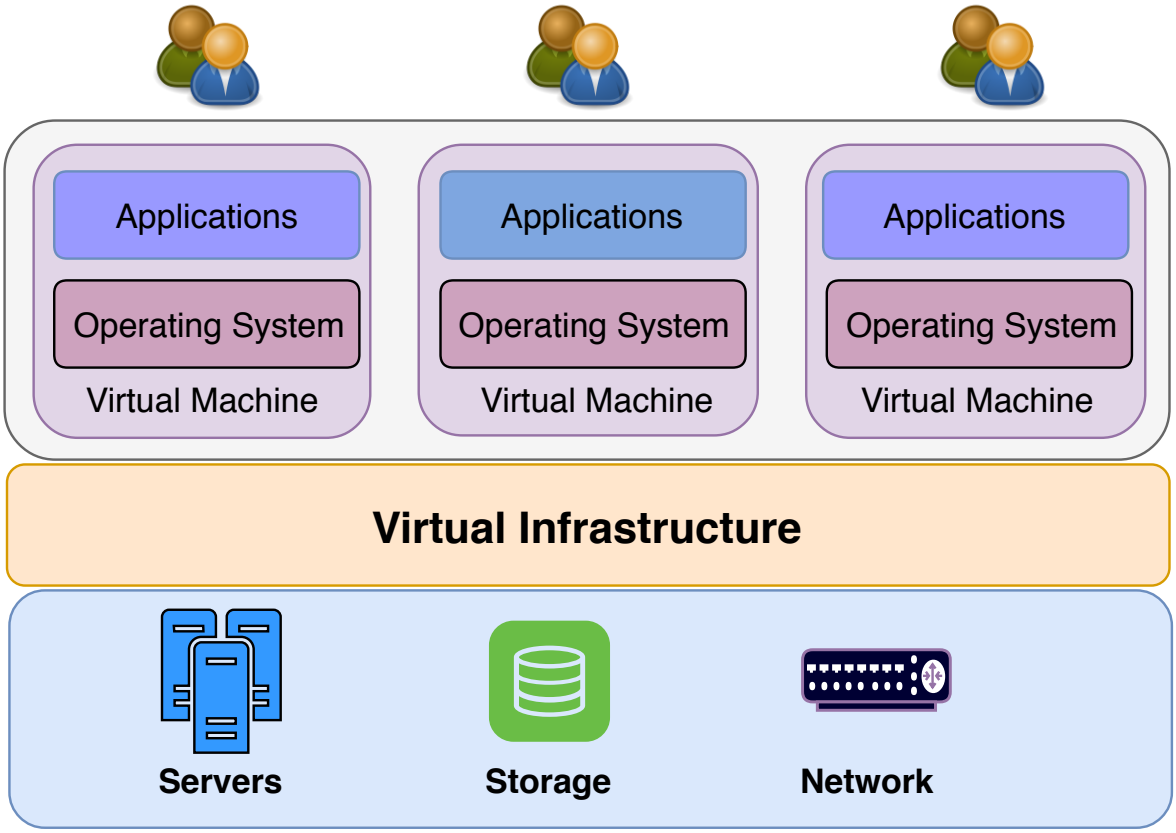


Figure 2.6: Mutli-Tenant Virtualised Environment

(NFV) aims at replacing hardware-based equipment with software-based network functions (NFs). It enables implementing and running NFs on off-the-shelf servers by using commodity programming languages, frameworks and virtualisation techniques. Therefore, NFV offers faster deployment and provisioning of service functions and addresses the problem of compatibility of vendor-specific hardware and reduces the capital and operational expenditure associated with them [70, 71].

NFV introduces the benefits of software-based solutions to security systems. NFV offers cost reduction, solving compatibility and updating issues. Software solutions are inexpensive compared to hardware appliances as they eliminate the cost of the periodical rebuild or upgrade of the security system and the cost of maintaining vendor-specific knowledge. Pure software solutions can also benefit vendors: they allow them to put more effort into reducing the complexity of managing and re-configuring their products by providing easy to use programming interfaces. Furthermore, updating or upgrading software services is a matter of dynamically retrieving the new source code of software components rather than extending or

replacing hardware equipment. VNFs can be developed and run on commodity x86 servers. It is usually encapsulated in VMs or lightweight containers to lower the hardware requirements and increase the NF-to-host ratio [62, 71–73]. VNFs can be started and teared-down in significantly less time compared to weeks (the time it takes to design, purchase and deploy a new middlebox in a traditional network).

While VNFs provide many benefits for security systems, it is worth mentioning the challenges they face. For instance, the performance properties of generic software NFs are inferior to their hardware counterparts, since general-purpose hardware and software have originally not been designed for high-speed packet processing. In addition to, the need to use multiple CPU cores to achieve line rate processing with the rise in capacity of network links [74]. In order to tackle the performance challenges without sacrificing deployability, many research projects are focusing on new approaches to address these challenges. For example, open-source packet processing techniques (e.g., the Intel Data Plane Development Kit [7]). Another challenging aspect is performance isolation between network services sharing the same physical hardware such as the work in [74] to improve both efficiency and fairness in sharing resources among software middleboxes.

2.4 Virtual Network Functions Orchestration

Implementing security functions as VNFs in a multi-tenant virtualised environment will mitigate the hardware middleboxes problem and offer the benefits of software middleboxes as discussed in the previous section and summarised in Figure 2.7. Moreover, it introduces the efficiency of cloud services to the network function deployment process and offers the customisation for the multi-tenant environment.

However, many design issues need to be tackled when managing security VNFs in multi-tenant environment such as where are the locations that VNFs will be allocated in the infrastructure, which location to select for a new VNF request, which VNF to migrate when a

[7] https://dpdk.org

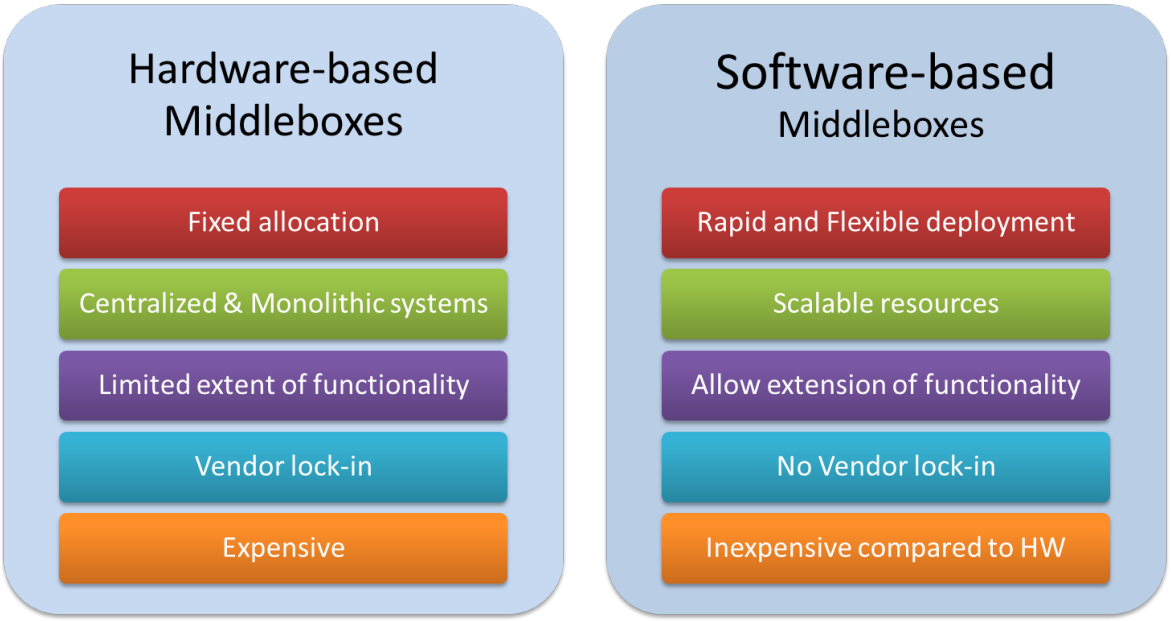


Figure 2.7: Hardware vs Software Middleboxes

server is overloaded, what are the mapping strategy of services requests to VNF instance(e.g. 1:1 mapping each services request is deployed to one VNF instance), are VNFs instances shared among multiple tenants to save resources or non-sharing policy is adopted. Such decisions must be carefully designed not to turnover the gains of virtualised services as they highly affect the network performance and can result in resource wastage, or cause bottlenecks [3]. For example, where the functions will be deployed and how the data center routing will be affected, as in some cases redirecting traffic to hosts that are not always on the shortest path will increase experienced end-to-end latency undoubtedly.

In this section, we explore how research addresses the challenge of deploying Software middleboxes as VNFs in a virtualised multi-tenant environment. Designing new tools to manage network functions in a virtualised environment is an active research area where many researchers propose different answers to aforementioned questions. Recently many of these approaches exploit Software Defined Networks (SDNs) features to introduce dynamism in managing VNFs as the work proposed in this thesis. Thus, we start by depicting the SDN architecture and its characteristics below.

2.4.1 SDN

2.4.1.1 Architecture

Software Defined Networking (SDN) promotes the decoupling of data and control planes of the network. Driving by the powerful network architecture that SDN can provide for, the Open Networking Foundation (ONF) was formed by major companies like Google, Microsoft, and Facebook to endorse SDN through supporting OpenFlow [75]. ONF offers a high-level architecture for SDN that divides the architecture to three layers: infrastructure layer, a control layer and application layer [76, 77] as shown in Figure 2.8. The infrastructure layer also known as the data plane consists of network/forwarding elements such as physical or virtual switches that are connected to the network through an open interface. The control layer also known as the control plane is a set of one or more controller that orchestrates the forwarding of the data plan layer through an open interface. Application layer consists of user and business applications such as network services, security, analytics and network management application that communicate with SDN control [77]. Two main open interfaces that controllers use to interact with the other layers: the northbound API (e.g. REST API) to communicate with the application, the southbound (e.g. OpenFlow) to communicate with the forwarding devices. This separation and abstraction between the layers allow for new network control services to be implemented without changes in the underlying infrastructure which introduces programmability to networks [75]. Besides, SDN increases manageability, scalability, and dynamism of the network which enhances the capability of the system to handle security challenges [22, 58, 71, 76]. We introduce some of the characteristics that SDN enabled networks have that can be exploited to solve security challenges in the multi-tenant environment.

2.4.1.2 Characteristics

- *Centralised control:* In SDN, forwarding elements are directly connected to and controlled by controller software (e.g., Ryu or OpenDaylight). This centralisation of the

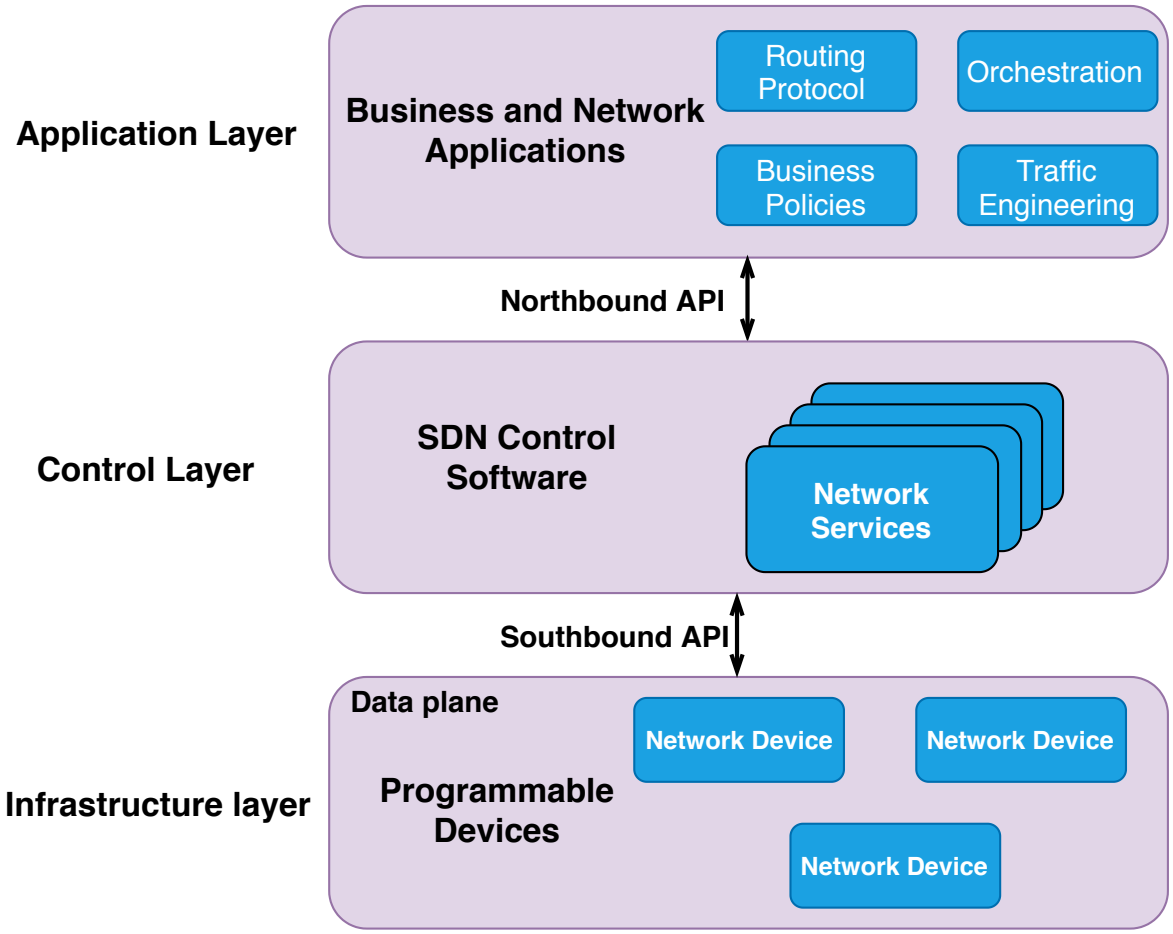


Figure 2.8: Software Defined Networking (SDN) Architecture.Source: SDxCentral

control plane enables a defence system to rapidly respond to network changes from a central controller through updating the forwarding rules of the entire network infrastructure. For example, custom policies can be applied through the controller instead of configuring each component separately [76].

- *Programmability:* The ability to apply custom routing policies in SDN through programming the controller, instead of by statically configuring each network element individually, introduces programmability to networks. Combined with the centralised control of the network, SDN provides the ability to programmatically steer traffic through network services hosted at any physical location of the network. Additionally, it can improve the efficiency of a security system through the dynamic control of traffic to achieve load balancing between security functions. Furthermore, programmability introduces dynamism that leverages the capabilities of an attack defence system

to mitigate attacks through automatic updates of forwarding rule as a response to attack detected. Also, SDN enables experimentation and testing new ideas by allowing inexpensive network management [22].

- *Global view of the network:* In contrast to a traditional network, in an SDN environment, the controller is able to maintain a global view of the network status and operation. The controller can query all the flow entries across the network to identify individual traffic paths, request per-switch statistics of the ports as well as flow utilisation. Furthermore, the controller can build a full topological representation of the network allowing (re-)routing decisions to be made. Combining all the available data at the controller, it is possible to have a fine-grained view of the network-level utilisation as well as identifying the flows, ports and hosts responsible for the bulk of the traffic. Using this information can increase a security system's ability to monitor and analyse network behaviour and reconfigure the network in response to changes.

2.4.1.3 SDN and Software Network Functions

There has been considerable research on using SDN for software network function management. However, most approaches only focus on network management such as advocate the use of SDN to the problem of per-flow steering [78] which explore the centralised controller paradigm to steer traffic to different network services/functions including security to enforce system policies and/or network functions chaining [55, 60]. Stratos presented by [3] is a network-aware orchestration layer for virtualised middleboxes. In addition to enforcing network policies through chaining, it uses SDN to dynamically instantiate new NF instances in response to workload changes. In [79], Tajiki et al. propose a resource allocation architecture which enables energy-aware service deployment for SDN-based networks, it targets the optimisation of power consumption while considering delay, link utilisation, server utilisation as constraints. In contrast to this approach, we advocate the use of SDN not only for flexible traffic steering but also as the underlying mechanism to dynamically distribute network functions where and when required across the network.

2.4.2 Orchestration Frameworks

There has been a considerable amount of work in software middleboxes and VNFs management and orchestration. Some of them, mostly earlier work, are specific frameworks that focus on particular aspects of an overarching architecture while others are more general frameworks.

2.4.2.1 Specific Framework

The ClickOS project proposed in [61] focuses on high-performance data plane NFs by reducing latency for packets that go through multiple NFs in the same location. It is named after the Click modular router [80] as it is used as the underlying packet processor. However, it does not have any network-wide control over the functions. Another example, in [81], the Slick programming framework is proposed to manage fine-grained functions that can be shared and composed into more complex packet processing sequences, also Slick elements can be allocated at arbitrary locations and traffic can be steered through them. OpenNF in [82] is a centralised control plane that orchestrates NF dynamically. It supports scaling through duplication and proposes coping the internal states and network forwarding states instead of copying the whole virtual machine to reduce migration cost. It utilises SDN to redirecting flows; however, this implementation involves a significant modification to the middleboxes implementations. FlowTags in [83] is another control plan for NF that is using SDN controller to dynamically steer traffic to enforce chaining and also tag the packets to guarantee the correctness of policy enforcement in case of NF mangling where functions may change packets header. While such platforms offer high-performance network functions, they use a custom hypervisor and restrict users to a specific programming language. However, a more dynamic system should utilise generic, widely deployable NFs.

2.4.2.2 General Framework

On the other side, a lot of recent research is targeted towards a sophisticated management and orchestration framework for NFV to solve what is known to be the VNF Orchestra-

tion Problem (VNF-OP). Others use the term VNF management, automation and orchestration (MANO) to refer to the same problem. Many Standard organisations, with the support of service providers, start projects to standardise the IT virtualisation technologies and the VNF orchestration in particular. The Open Linux foundation standard for VNF has the OPNFV Framework project [84]. They target creating a NFV platform reference to accelerate the enterprise and service provider transformation to the new technology. The European Telecommunications Standards Institute (ETSI) also has their own Open Source MANO [85] project. In Figure 2.9, we show the MANO framework proposed by ETSI. It consists of 3 main components. 1)NFV Orchestrator (NFVO): responsible for accepting new requests for Network Services (NS); monitoring and managing their life-cycle such as instantiation, scaling, performance measurements and termination; global resource management of NFV infrastructure resource requests; policy management for network services. 2)VNF Manager (VNFM): reasonable for the life-cycle management of VNF instances like NFVO for NS. 3)Virtualised Infrastructure Manager (VIM): responsible for controlling and managing the NFVI computing, storage and network resources such as orchestrating the allocation/upgrade/release/reclamation of NFVI resources. It is also responsible for optimisation of such resources usage, in addition to, collection and forwarding of performance measurements and events [86]. Some Researchers propose an implementation for the standard frameworks. For example, authors in [87] propose a policy-based MANO framework to orchestrate NFV services in SDN networks. The architecture addresses VNF life cycle management and service chaining for the Content Delivery Network (CDN), while no placement strategy is reported.

2.4.3 VNF Placement

In Enterprise networks, an orchestration system adopts a placement strategy that decides where an NF will be allocated. It considers network traffic, resources utilisation, and the multiplexing of different services and physical machines in a complex optimisation problem that initiated many research projects and considered as the main focus of this thesis. The problem is considered as one of the challenges that network providers face in their

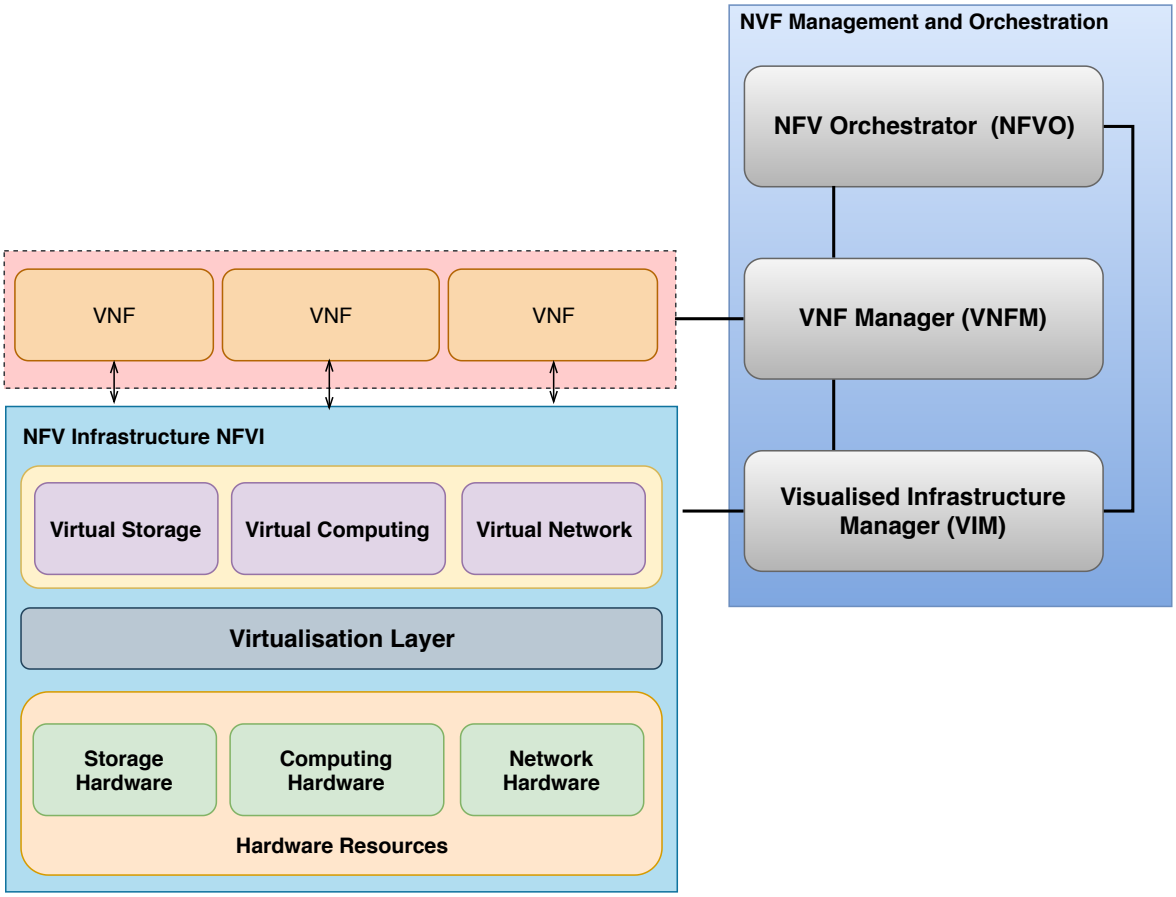


Figure 2.9: ETSI NFV Architecture Framework

transformation to VNF technology, where designing a placement strategies that minimise provisioning and operation cost is not trivial [88]. The objective of the placement optimisation problem can combine computing resources, power consumption and/or communication cost. Also, the optimisation problem can consider parameters such as SLA, policy rules, and/or security rules. While many researchers study the problem from different perspectives many propose a mathematical model for the problem. However, due to the complexity and domain of the problem, approximation algorithms are usually offered as a solution to the NP-hard problem. Designing a placement strategy has two approaches: design a static initial placement algorithm that consider a group of requested VNF to be allocated on distributed location with limited capacity for hosts and links, and a dynamic placement where an initial placement is required plus as a migration algorithm which selects one or more of the deployed VNFs to be migrated to optimise performance or solve a performance problem such as server overloaded, scaling up, not enough resources to accommodate new requests. In the

following, we discuss some of the recent and most cited work that has been done in the VNF placement problem.

Most of the earlier research done in the VNF placement is based on the work under virtual networks embedding (VNE) or VM placement (VMP). VNE is the mapping of a set of logical graphs of interconnected VMs on a substrate graph of shared physical infrastructure [89, 90]. Considerable research has been done on VNF placement as an instance of the VNE problem [91–94]. However, many researchers address the difference between VNE and VNF placement such as flow demand representation and absence of chaining in VNE problems [90, 95–97]. They also point out that VNF placement is a considerably harder problem to solve than traditional VNE problems.

VMP is the process of selecting which VMs should be allocated at each physical machine (PM). Also, many researchers address the problem of VNF placement as an instance of VMP problem [98–102]. However, a technical paper by Grochowski from Juniper Networks [103] claimed that allocating VM requests by current cloud scheduler (e.g. OpenStack) is based on metrics like available RAM, storage and compute which proven to enough for traditional cloud workloads as memory usually is the number one in contention for web-based apps and services, then come CPU and storage resources but not bandwidth. Also, in the case of common workload, per-node bandwidth utilisation has never been a problem that needs to be addressed. However, in the case of purpose-built VNFs, the requirements for CPU, storage, and memory are almost fixed, but the variation of network requirements for each user is significant. Bouet et al. in [104] also discuss the differences between the two problems, they argue that VM placement is a node-centric problem where many VMs exist as small end-points, while a VNF placement is a network-centric problem where few VNF exist as large middle-points. The VNF placement problem is prevalent with dozens of papers trying to tackle it from different angles in the last few years, we discuss some of these approaches below.

While VNF placement problem depends on several parameters such as computing cost, link bandwidth, QoS, economic profit, network load, energy efficiency, security.. etc [105]. Most

of the proposed approaches consider either one or two of these parameter as objectives which is occasionally are contradictory objectives while other parameters to be considered as constraints. MORSA in [106] provides an NFV infrastructure with a multi-objective approximation genetic algorithm for resource scheduling that considers computing and communication cost. In [107] Luizelli et al. focus on the servers cost in large infrastructures by using objective function to minimise the number of VNF instances with end-to-end latency constraints. Palkar et al. in [101] introduce E2, a scheduling framework for VM-based VNF. It aims to minimise intra-server traffic when mapping VNFs, it decomposes NFs to reuse some I/O operations functions such as TCP reconstruction, but not to reuse core processing blocks. E2 estimate the number of instances of each VNF based on traffic load the per-instance capacity and uses dynamic scaling to save resource by using latency threshold to detect overloaded VNF, however, it designed for specific architecture where VNFs are interconnected by a Layer-2 network. Wen et al. in [108] solve the Network Function Consolidation (NFC) problem targeting minimising the number of deployed VNFs. In [109] Kuo et al. identify Joint VNF Placement and Path Selection (JVP) Problem to allocate services chains as shared VMs with capacity constraint, the authors considers the relation between the link and server usage.

Furthermore, Cohen et al. in [110] minimises the system cost which is divided into setup cost and distance cost to reduce the distance between clients and services. They reduced the VNF placement problem into two NP-hard sub-problems facility location problems and the generalised assignment problem. [111] proposed SAMA a sampling-based Markov approximation algorithm which was used to find an efficient solution to save functional costs and network traffic costs. Hsieh et al. in [112] proposed a network-aware service that minimises server cost. Eramo et al. in [113] target reducing the energy consumption using a consolidation algorithm based on a migration policy of chained VNFs. In [114], Qu et al. formulate the VNF placement problem as a series of scheduling decisions aiming at minimising the latency of VNF scheduling by assigning the execution time slots to different services traversing the same VNF. In [102], authors consider the problem of VNFs placement for minimising the end-to-end delay with deployment cost and SLA constraints in multi-cloud

scenario. In [115], authors' goal is to speed the placement solving by narrowing the search space of the VNF placement. It restricts locations of VNFs to what they called the accessible scope, which is the number of servers close to ingress or egress nodes of the VNF flow. The results explore the optimal size of accessible scope through extensive experiments. The approach reduces non-shortest path routing to some extent while enhancing the time efficiency of the placement. However, it assumes functions are packet or flow-based. In [116], Pei et al. explore VNF management for SDN/NFV-enabled networks. It uses Reinforcement Learning to forecast future requests to reduce setup latency. However, it assumes functions are based on packet or flow and offers to duplicate VNF for load balancing.

2.5 Security VNF Challenges

2.5.1 Security VNFs Potentials

Many challenges need to be addressed for security VNFs. Yet, implementing security network functions such as firewalls and IDSes in software has the potential to increase efficiency and flexibility of a defence system for cloud environments [3, 61, 117]. Some of these potentials are detailed below:

2.5.1.1 Efficient Resource Provisioning

The rapid and easy deployment of VNFs increases the system's flexibility to react to changes such as traffic dynamics, dynamic resource (e.g. VM) allocation or the adding of new security functions. Therefore, it increases the efficiency of the system to handle attacks and maintain a consistent security policy. VNFs also offer dynamic up and down-scaling on-demand, leveraging the system's ability to handle traffic changes and attacks and at the same time maintain efficient management of resources which is considered a more efficient approach to the fixed under-utilised resources of hardware middleboxes.

2.5.1.2 Modularity and Chaining of NFs

As VNFs are implemented in software, they allow effective modularisation of security services and small component reuse to build more complex and customised security systems. The modularisation encourages developers and vendors to focus on building more efficient, but standalone modules instead of large monolithic applications. As a concrete example presenting modularity, one could build a high-performance IDP (Intrusion Detection and Prevention) NF by using a high-performance packet processing library (e.g., Intel DPDK), a software switch (e.g., Open vSwitch [118]) and an open-source IDP software (e.g., Bro). Moreover, modularisation allows the chaining of NFs to apply complex security policies. As an example, a common service chain consists of packet classifiers and firewalls or IDPs functions that are only used for a specific set of traffic (identified by the packet classifiers).

2.5.2 Customised Security Services

Furthermore, implementing softwarised security functions as VNFs support providing customised security services to users of multi-tenant environments. As tenants in multi-tenant virtualised environments run different applications, they require different levels of security per application. For example, a web server may require protection against HTTP flooding attacks, while critical servers may require deep packet inspection and/or a combination of signature-based and anomaly-based intrusion detection. As hardware appliances are designed to process all traffic passing through with very limited capabilities to specify different operations on specific parts of the traffic, there is no opportunity to specify different services for different users when using hardware middleboxes [7, 119]. In multi-tenant virtualised environments like cloud, security services are offered by Cloud Services Providers or third-party companies to satisfy the need for customised security services [3]. These services offer to deploy and manage virtualised security solutions to ICT organisations, and there are becoming more efficient and cost-effective than to hire a team of specialists to deploy and run your own security tools. With new security threats appearing every day, more of these services are emerging.

For example, Amazon's AWS security services have increased over the past few years. It starts with tools to monitor and control access to your application, such as CloudWatch and CloudFront. In 2015 it offered Firewall web application (WAF) to AWS users and Amazon Inspector which is a Host-based signature IDS with a knowledge base of hundreds of rules mapped to common security best practices and vulnerability definitions, These rules are regularly updated by AWS security researchers. They also demonstrate how using their monitoring and scaling services such as (CloudWatch, CloudFront and Autoscaling) can detect and mitigate DDoS attacks. Later in 2016, they announced their first DDoS Protection service a.k. AWS Shield, which is a managed DDoS protection service that safeguards web applications running on AWS. It detects, logs, reports and response to threats. In 2017, GuardDuty was announced as an intelligent threat detection service that analyses billions of events from multiple AWS log sources. It uses threat intelligence feeds, such as lists of malicious IPs and domains, and machine learning to detect threats more accurately. There are also third-party services that are available now from many security companies such as Alert Logic Cloud Defender, Armor, CISCO, Fortinet which provide services that are compatible with most cloud providers such as Amazon, Google and Microsoft.

2.5.3 Security Functions Orchestration

There has been considerable research on managing security network function. For example, the work used SDN such as Yoon et al. in [75] propose managing routing to security functions such as firewalls and IPSs from a Floodlight SDN controller through a designated SDN application. The application can update flow rules to forward traffic to the appropriate security function that is connected to a specific interface of the switch. Another related example of utilising SDN for improving security is presented in [120]. In this work, the authors distribute security functions between switches and an SDN controller. Specifically, a local detection component is installed on each SDN switch, and a global detection component is installed on the controller to detect attacks that can only be seen on the global view. Implementing security functions on switches and/or controllers introduces scalability issues

where resources are limited to detect intrusions in the case of high volume attacks or traffic changes.

Furthermore, few systems were proposed to provide customised security services to users in a multi-tenant environment. In [121], a framework to provide service-based intrusion detection to the cloud was proposed. Different services are provided as subsets of the Snort signatures database. User subscription to a service means testing the user traffic by the subset of the signatures corresponding to the service. Subscription management is implemented through a web-based application which allows each user to add or remove subscriptions or view alerts. The system provided centralised signature-based protection with all traffic having to pass by to be processed. Roschke et al. in [122] propose an IDS management architecture for the cloud. Each VM is secured by host-based and network-based IDS implemented as virtual machines on the same physical server. Each IDS can be reconfigured by the cloud users to meet their needs, such as dropping unused rules or changing threshold parameters. A central management system is responsible for correlating alerts from all IDS. No implementation for the system is discussed. The previously proposed systems suffer from flexibility and scalability problems: such as, limited services, inflexible deployment location (e.g. the same server as the user) and/or a central processing point that leaks scalability and resilience.

Still, only a few researchers consider the distinct requirements and constraints related to security functions in the placement strategy implemented. The authors in [7], address the allocation of security services in virtualised environments and discuss their challenges. They model the allocation problem for ISP networks to minimise the cost of operators as a Mixed-Integer Linear Programming (MILP) problem, but no implementation is reported. Both approaches only consider security functions that process traffic at the flow level. While Bouet et al. in [104] address the Cost-based placement of vDPI functions in NFV infrastructures with the objective to minimise the number of licenses used as their cost function.

On the other hand, in [123] authors examine a placement strategy to provide a multi-function multi-tenant NFaaS. They suggest two strategies tenant-centric and service-centric. Tenant-centric where all VNFs leased by a tenant are mapped into a single server which proved

to save physical bandwidth and reduces network delay while service-centric maps VNFs of executing the same service for different tenants on the same server which proved to have better resource utilisation. However, this approach considers a VM implementation inside cloud servers to the VNF which introduces communication overhead of rerouting traffic to the NF hosts. In [124], Shameli-Sendi et al. propose a model for network security defence patterns (NSDP) to advocate the best practice of deploying security functions in a multi-tenant environment. The model considers security constraints such as placing an IDS on an encrypted channel and steering all flows of a tenant go through a single point to be tested by a specific function. However, the proposed optimisation algorithm considers flows of the east-west traffic in the network. While in [91], Dwiardhika and Tachibana approach security from a different perspective where a virtual network is mapped to a substrate network if the security level matches. At the same time, the placement algorithm is allowed to place more security VNF to increase the substrate security level to match the virtual network demand, which represents a minimum cost. The VNF placement is treated as an optimisation problem with genetic algorithm solution. However, security VNF functions are shared among mapped networks.

2.5.4 Current Issues and Limitations

As the use of virtualised middleboxes becomes more widespread, research is focusing on the different aspects of managing network functions in virtualised environments. Nevertheless, only a few consider the distinct requirements and constraints related to security functions and multi-tenant environment, which are the main focus of this thesis. We discuss the main differences below.

2.5.4.1 Shared Security Modules

Sharing security modules among multiple tenants that has been adopted in previous work [3, 91, 104, 107–109, 113, 114, 121], to save resources or to reduce the number of licences, is not feasible. It does not allow for customisation required by multi-tenant users and intro-

duces some security risks. Some proposals implemented virtualised functions similar to their hardware counterparts, i.e., as high-speed high-capacity appliances. They process traffic for different users in the network within their capacity where it is assumed that sharing security modules among tenants could save some static resources such as rule set memory space or the modules binary code space. However, key configuration differences may exist based on different security policies and strategies among different tenants which make configuring shared instances to keep the consistency of configuration (e.g. rule set) is a very complicated process. Besides, increasing the size of the rule set to include specific rules for each tenant where needed will increase the processing time and resources and decrease the throughput.

Furthermore, this process must be handled only by the framework, not the tenants and probably manually where no tools exist to handle this kind of configuration that depends mainly on each module which leaves tenants with no direct access to control their modules except through the framework and therefore cancel any gain from the sharing process. Furthermore, specific security functionality that is based on building a behavioural model for traffic (e.g. anomaly detection modules, cannot be shared since each tenant will have a different normal behavioural model. Besides, tenants should be allowed to configure their security functions themselves. Besides, if the same binaries are shared among different tenants, then access control becomes cumbersome, and there are real risks for illegitimate access that can cause security policy violations. Furthermore, while most optimisation models assume that VNF of the same function is identical in their resource consumption and throughput, However, Service Providers offer the network function with different configurations to guarantee different levels of quality of service such as different throughput based on the number of used CPU cores [125]. Thus, in multi-tenant environments where services are offered on a per-tenant basis, sharing security functions among different tenants will increase the complexity of managing them and reduce the ability to have customised services to fit different tenant needs.

2.5.4.2 Security-specific Traffic Constraints

Security functions have more traffic constraints in the allocation than other network functions. Network functions use extracted features from the processed traffic as input data for their operation, as shown in Section 2.2.4.3. Simple features can be extracted directly from packet header (e.g. IP address, port numbers, transport layer protocol) while other features need to re-construct flow or aggregate more than one flow. Due to the complexity and the different forms of attacks today, security functions use more complex features compared to other functions (e.g.NAT, WAN optimiser). For example, DDoS attacks on a web-server can be detected by observing features (e.g. request rate, download rate, uptime, page access rate) [53]. Because traffic in data centers is usually split on multiple paths due to Equal Cost Multiple Path (ECMP) routing, capturing representative traffic of complex features is not a trivial process. It involves rerouting some of the network traffic based on the implemented placement algorithm, which can affect the network performance metric such as delay and throughput. Which are not presented in other network functions where they are mainly based on features of a single connection or flow and addressed in previous work [3, 7, 115, 116]. Consequently, security functions in data centers hold more traffic constraints in the allocation than other functions.

2.5.4.3 Duplicating Security VNFs

Duplicating VNF in case of security functions is limited where many approaches used duplicated instances of a security function as a solution to function overload or split of traffic [82, 116]. This is applicable for security functions that work on a packet or single flow level, however, for an anomaly detector to capture an accurate behaviour model that is based flow aggregation features in ECMP networks, the intended monitored flows must all be steered to one instance of this type and it cannot be duplicated in case of traffic split. This approach has been widely adopted and involves all the data continuously pushed to a central point for analysis [126]. Others propose that computational, transmission and storage cost of monitoring traffic for anomaly detection can be reduced by mapping the traffic

to less dimensional space by extracting features and only transmit these features in case of distributed nodes monitoring [127–131]. In [128], Huang et al. aim at increasing scalability for networks with a large number of monitored nodes. The proposed approach is based on distributed tracking combined with approximate PCA analysis where the architecture involves a set of local monitors that send quantised data streams to a coordinator which makes global decisions based on these data. In [129], Keralapura et al. use a threshold count to minimise the communication overhead between communicating nodes in a wireless sensor network. They accomplish that by aggregating the frequency count of an event which is continuously monitored by distributed nodes and sent to a centralised point whenever the actual count exceeds a given threshold to detect anomalies, however, this approach only considers hardware-based security middleboxes.

2.5.4.4 Traffic Direction

Traffic inside data centers flow in two main directions, ”North-South” which is traffic moved in and out of the data center, while ”East-West” is traffic internally generated and consumed by various applications' instances within the data center where multi-tier is the dominant paradigm for enterprise application. East-west traffic is the dominant flow in data centers [132] with reports that it will reach 85% of total traffic by 2021 [133], although north-south traffic is usually what produces east-west traffic. For instance, a user request to a web-based application hosted in the cloud can result in multiple requests to application servers in the form of internal communication will occur, before the final answer to the original request has been found and sent back to the user. However, north-south traffic is what most data centers security solutions middleboxes are focusing on (e.g. Firewalls and IDs) as they are not designed for east-west traffic and mainly designed to protect users from outsiders attacks. Moreover, east-west traffic is based on encrypted communication between the application instances/tiers and to find internal threats, application-based security solutions must be designed to decrypt messages and to analyse communication protocols between application instances. Furthermore, east-west traffic is always content in TOR level to reduce

contention. On the other hand, most previous approaches of orchestrating network function consider east-west traffic of the network [101, 111, 112, 124]. They argue that it is the dominant traffic direction; however, this is not the case when dealing with security functions as it is designed for north-south traffic.

2.6 Summary

This chapter studies the virtualised security network function in a multi-tenant environment and analysis related work of orchestrating VNF and security. To comprehend the background of virtualised security middleboxes, an overview of network threats and their classification with DDoS attack as an example, along with recently reported attacks and their impact. Security solutions such as firewalls and IDS have been discussed along with a demonstration of the complexity of traffic processing in their operation. Legacy hardware middleboxes have been discussed and their challenges in a multi-tenant virtualised environment. As it suffers from lack of deployment flexibility; expensiveness; and limited extension of functionality and the inefficient management of resources, software middleboxes and VNF were introduced, and their potentials for security were demonstrated. However, orchestrating virtual network function is a complex process that inefficient one can turn over the outcome of the virtualisation. An analysis of orchestration frameworks has been followed by a review of the latest state of art search on the placement part of the orchestration. It has demonstrated the limitations of currently proposed systems of deploying security functions in multi-tenant environments due to the unique constraint and requirements of these functions such as the complexity of features extracted from network traffic and their processing to north-south traffic is contradictory to other network functions. Based on the previous discussion, this thesis will address the efficient placement of security VNF in a multi-tenant virtualised infrastructure. In the next chapter, we propose a placement framework of the security functions in a virtualised infrastructure. We identify the placement constraints then propose a classification to guide the placement strategy of the framework to efficiently allocate the security functions.

Chapter 3

Design of a Resource-Aware, Security Placement Framework

3.1 Overview

In multi-tenant virtualised data centers, users run different applications, each with different security requirements. For instance, a typical web server may require security modules to detect and mitigate HTTP flood attacks and SQL injections, while critical servers may require a firewall, IDS and/or DPI to guarantee high availability and data integrity. We propose a *placement framework* for security services in multi-tenant virtualised data centers where security modules are deployed as VNFs throughout the infrastructure. Based on the classification of how modules process traffic, we identify the allocation strategies suitable for allocating security functions through the network infrastructure. The framework offers the customisation to fulfil diverse tenants needs for different security services and levels of protection and reduce the complexity of having to manage shared modules. A placement algorithm is responsible for selecting allocations for security VNFs to satisfy the requirements and constraints of the security service requests. We propose a resource-aware placement algorithm that maintains an efficient usage of the data centers resources by maintaining minimum usage of computing and networking resources.

The following sections present the design of the proposed placement framework. The framework architecture and its characteristics are presented in Section 3.2. In Section 3.3, we propose a security module classification based on traffic processing granularity. Then, we design allocation strategies for the resource-aware placement framework that are based on the classification. Then, we represent the constraints and objectives of the placement. In Section 3.4, we formulate the placement problem then show the reduction of the placement problem to the variable size variable cost bin-packing problem. Finally, We adopt heuristic, meta-heuristic and near-optimal solutions to the placement problem, and we propose a one-dimensional implementation to the placement that eliminates the communication overhead of the problem in Section 3.5.

3.2 Framework

We propose a security placement framework for multi-tenant virtualised environments that address the challenges of legacy and monolithic security deployment through exploit SDN and VNF technologies. *Security Services* will be offered as software modules that are allocated throughout the infrastructure as VNF to process the designated traffic. The modules are offered on a per-tenant basis, and each request will result in allocating a module which will be deployed as one or more VNF instances of the software module to process flows of the requesting tenant. A high-level architecture of such the framework is presented in Figure 3.1.

3.2.1 Architecture

The framework is responsible for the placement and management of security services. As shown in Figure 3.1, it is managed from a logically centralised controller that maintains a network-wide view and handles communication to and from the network infrastructure. The framework stores security software modules in a database that can be easily updated. It monitors the system components, their temporal resource utilisation, the network state (e.g.,

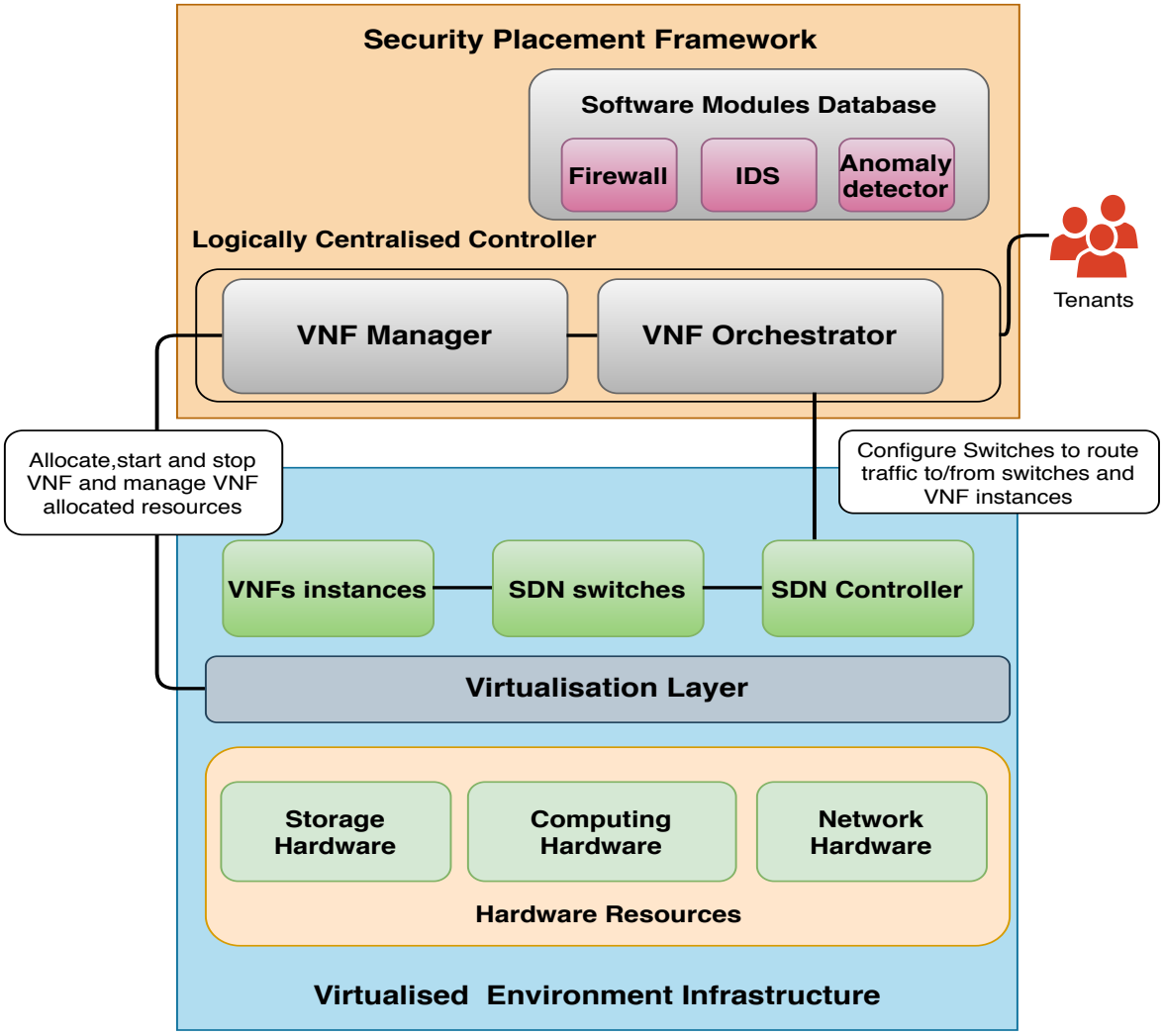


Figure 3.1: Security Placement Framework

traffic distribution, network failures, VM migrations), and subsequently responds to reflect any operational changes. The controller consists of two main units: a VNF orchestration unit and a VNF manager unit. The VNF orchestration unit is responsible for accepting tenants' requests for security services and converting it into deployment orders of VNF instances. It is also responsible for the placement process where it selects the locations of the VNF instances which are then executed by the VNF manager unit.

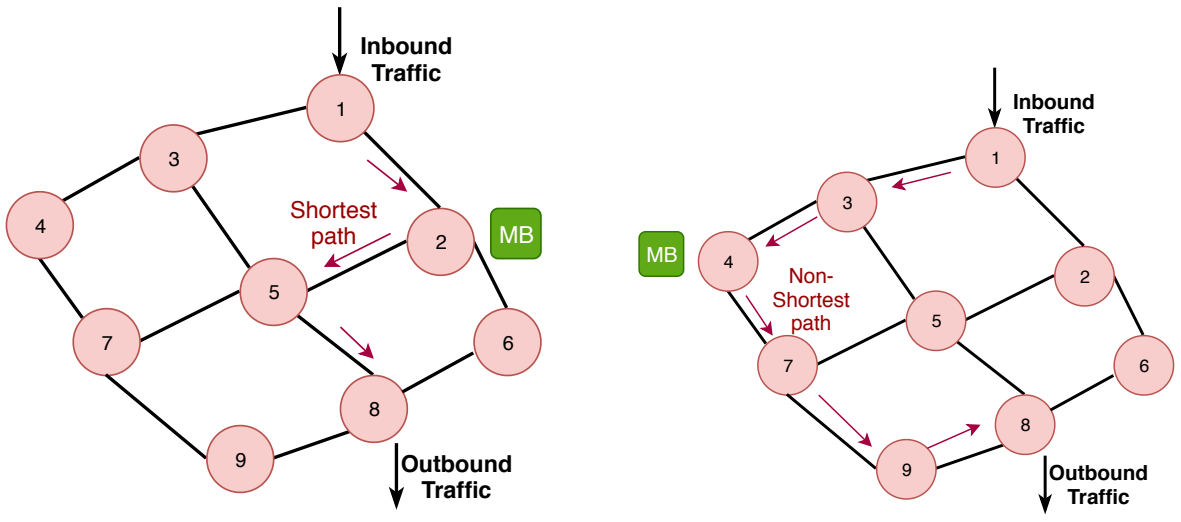
Furthermore, it communicates with the SDN controller to install the rules to enforce steering the designated traffic through switches to instances' selected locations, and periodically retrieving flow and port statistics from all network devices. While the VNF manager unit is responsible for managing the virtualised network and compute resources available for the

security functions deployment. It is responsible for the start and stop of the VNF instances and allocating their resources with the capability of provisioning their resources if needed by continuously monitoring their performance. Furthermore, it reports periodical updates of locations resources to the VNF orchestration unit. The main characteristics of this system are detailed below.

3.2.2 Characteristics

3.2.2.1 On-path Deployment

Our approach allocates software modules as VNFs on-path of the actual traffic to avoid redirecting traffic through a non-minimal path. This approach will reduce the detour cost of the path that traffic has to take to pass through a security function as shown in Figure 3.2. Figure 3.2a shows traffic flow from inbound point 1 to outbound point 8 with the shortest path through 2 and 5. On-path deployment will require the middlebox to be deployed on the shortest path for instance at point 2 as shown in the figure. The off-path deployment will not restrict deployment to shortest path points as shown in 3.2b where middlebox deployed in point 4 that is not is the shortest path selected by the routing algorithm which requires traffic to be rerouted to a non-shortest path (through points 3,4,7 and 9).



(a) On-path Deployment

(b) Off-path Deployment

Figure 3.2: On-path and Off-path Deployment

To allow the on-path deployment of the security VNFs, deployment location must be on-path too. Therefore, the framework will utilise location that is collocated with switches of the network to deploy the security functions. Traffic is rerouted from switches to the security function and back as shown in Figure 3.3.

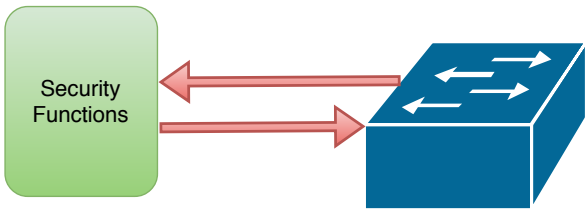


Figure 3.3: Security Functions Collocated with Network Switches

This approach will reduce overhead and save network bandwidth which complies with our resource-aware approach. However, It requires that the placement algorithm will be routing-aware to determine the on-path points for the placement of VNFs.

3.2.2.2 Traffic Directionality

While east-west traffic is the dominant traffic direction in data centers, security functions are designed to protect against north-west traffic as illustrated in Section 2.5.4.4. Furthermore, east-west traffic is concentrated on the top of rack level where data centers operators try to consolidate traffic-correlated VM (usually instances/tiers of the same application) in one rack to avoid congestion. Rerouting this traffic outside of top of rack switch for processing is a process called hair-pinning and can result in a massive explosion in the numbers of middleboxes and increases the complexity of the routing which pointed by VMware COO in [134] Therefore, our approach only considers north-south traffic when placing security function such as firewalls, intrusion detection and deep packet inspection.

3.2.2.3 Services to Security Functions Mapping

Our work reduces the complexity and security risks of deploying shared modules by adopting a non-sharing strategy. As pointed out in Section 2.5.4, sharing security modules among multiple tenants may save resources but increases the complexity of managing these services.

Thus, the non-sharing policy will eliminate this complexity and introduce the customisation that can be offered to a multi-tenant environment such as quality of services (e.g. throughput) by service providers. Furthermore, it reduces the security risks for illegitimate access that can cause security policy violations. Therefore, Our framework accepts tenants' requests to security services and maps these requests to security modules dedicated to processing the traffic of the requested tenant and deployed on-path of this traffic.

3.2.2.4 Elastic Security Provisioning

The framework implements security services as VNFs with logically centralised management to allocate, deploy, and orchestrate them in software, hence allowing for flexible scaling, reduced deployment time, and minimal reconfiguration overhead. The deployment flexibility provided by NFV allows the elastic deployment of security functionality when and where required, hence increasing resource usage efficiency. For example, a new (e.g., mitigation or filtering) function can be deployed in response to the detection of an attack, or new instances can be added to distribute attack detection and prevention to multiple points. Policies in general and security functionality, in particular, can also be migrated in response to a reconfiguration of the network or the services running on top of it (e.g., live VM migration or consolidation).

3.2.2.5 Service-based Model

NFV adheres to the service-based model of cloud computing. It offers the key features of any cloud service such as the abstraction of the infrastructure and applications as service interfaces, the sharing of multi-tenant resources, the on-demand self-service provisioning, and near real-time deployment. It allows each tenant to request security services according to the required level of protection and SLA. For instance, a tenant can require a firewall, IDS, and DPI services on top of their standard leased resources, or a combination of them with the possibility of changing the requested services later.

3.2.2.6 Resource-Aware Allocation

The allocation strategy for placing the security functions ensures capturing the indented traffic for accurate and efficient detection while incurring minimal impact on the monitored traffic/services through, e.g., maintaining shortest path routing and reduces the overhead of the framework. The allocation decision of a security software module is based on three factors.

1. Require minimum traffic steering.
2. Ensure enough resources are available to accommodate the additional module on the chosen location/host.
3. Efficient management of resources to reduce duplication, increase the network-wide security system usable capacity.

To achieve the resource-aware allocation, a *resource-aware* placement based on the security modules classification according to traffic processing granularity will be introduced in the next section.

3.3 Resource-Aware Placement

The placement of security modules VNFs in multi-tenant data centers can be defined as selecting a location for VNF instance(s) to fulfil a tenant's request for a security module. The placement must guarantee that the module is working correctly by satisfying the module constraints such as (the required traffic is passed to the module, enough computing resources are available at the location for the module to process the traffic...etc.). However, the resource cost of allocating the modules varies based on the location point(s) selected, for instance, one allocation may require traffic to be steered to a non-shortest path while another may require multiple instances to be deployed. Therefore, when many allocations satisfy a module's requirements, the placement algorithm is responsible for selecting one of them based on the

certain objectives which make it a resource allocation problem. A typical objective of VNFs placement can combine computing resources, power consumption and/or communication cost under specific problem constraints such as (quality of services and end-end-delay..etc). However, security modules process traffic differently from other network functions which affect the selected location for deployment. Thus, we begin by classifying security functions based on traffic processing granularity.

3.3.1 Traffic Processing-based Classification

Security modules process traffic in different ways, depending on how threats are being detected. Traffic can be processed on a per-packet, per-flow, or per-flow-aggregate basis where each level can protect against different types of security vulnerabilities. For example, per-packet or per-flow processing cannot detect threads that span multiple flows (e.g. DDoS flooding, worm spreading and probes). Therefore, each module requires a different granularity of traffic processing and traffic distribution in the network will constrain the allocation of security modules to locations where this granularity can be satisfied. We have produced a set of equivalence classes of security functions based on the detection method of different attacks and subsequently, the granularity of the traffic being processed, as illustrated in Figure 3.4.

3.3.1.1 Stateless

Stateless (packet-based) class: The first equivalence class represents modules that process traffic at the individual flow or packet level. Detection or mitigation decisions are made based on the state of a single packet or flow. The typical operation of a module of this class is to match patterns of the packet or flow specification against a database of signatures/access lists and take action on finding a match (e.g., block, pass, alert and/or log). Since this packet/flow matching on a given signature is done independently at different links, replicated instances of this class can be distributed across multiple network locations. This can be achieved by per-flow routing and by placing duplicate detection modules at diverse network locations

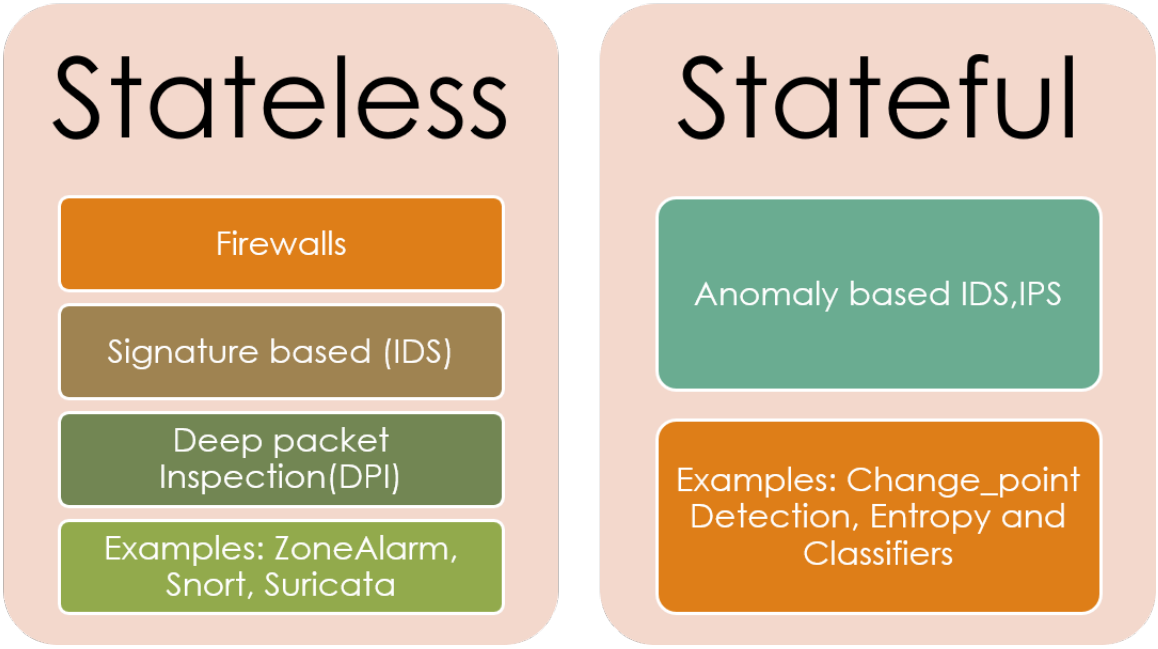


Figure 3.4: Security Function Equivalence Classes

where traffic matching a certain specification is being split due to ECMP routing. Examples of this equivalence class include Access Control List (ACL)-based stateless firewalls that evaluate packet contents statically; firewalls that keep track of the bidirectional state of network connections (e.g., TCP streams, UDP communication) [135]; signature-based IDS and Deep Packet Inspections (DPI) [136] systems where header/payload data are processed against a database of known attack signatures (e.g., Snort [13], Bro [40], Suricata [14], etc.).

3.3.1.2 Stateful

Stateful (flow-based) class: The second equivalence class consists of security modules that process traffic to extract anomalies based on coarser granularity than a single flow. They use techniques based on different features of traffic such as changes in traffic volume (Change Point Detection [137]), deviations in a given traffic feature distribution (Entropy, Histograms, etc.) [138], or use more complex machine learning techniques such as, Outlier detectors [46], classifiers, neural networks and SVM [15]. They mine information from flow aggregate features to construct a model of normal behaviour and detect anomalies based on deviations from such normality. Therefore, steering all the intended monitored flows to one instance of

this type results in an accurate construction of the behaviour model. However, some modules may be able to periodically share meta-information between distributed instances that work together which can be an alternative to steering the entire traffic flow to one instance.

3.3.2 Allocation Strategies

Based on the previous classification, we propose four resource-aware allocation strategies that can be adopted when placing a security network function on-path in the network infrastructure as pointed out in Section 3.2. These strategies are designed to 1) satisfy the traffic granularity of the function as represented by the function class and 2) save the network and computing resources of the infrastructure. The on-path allocation targets minimise resources overhead by the system, however, most modern networks support ECMP-based routing where traffic is split among multiple paths. Therefore, the proposed strategies must consider the ECMP traffic split in the network topology to ensure the capture of the intended traffic by the network function.

3.3.2.1 Independent Duplication

Security modules that process traffic based on flow or stateless packet-level characteristics (e.g., stateless detection based on packet header signatures) can employ replicated instances across traffic split switches without the need for further coordination between them as long as the entire traffic destined to a particular tenant is monitored. Independent duplication will adopt this strategy where independent replicates VNF of the same module will be deployed whenever traffic is split.

3.3.2.2 Dependent Duplication

Dependent duplication strategy will adopt the distributed approach as illustrated in 2.5.4.3 where modules that process traffic based on coarser granularity than a single flow such as (anomaly detection based on statistical properties of the aggregate traffic) need to coordinate

between duplicate instances to accurately capture the traffic features. Although, the communication overhead bandwidth resulting from the coordination between VNF the instances should not exceed the communication overhead in case of redirecting traffic to one instance.

3.3.2.3 Single Instance

Approaches that based on coordination between distributed instances propose a trade-off between the accuracy of anomaly detection and the amount of data communicated over the network which based on how much data reduction has of the communication overhead compared to one instance deployment has been achieved and how frequent the nodes need to share information to maintain accurate detection [128]. Therefore, single instance strategy will be adopted when coordination between instance is not acceptable or does not provide any gain such as (e.g a function based on many features to extracted which results in huge amount of information need to be shared that is equal or more than the original traffic or the information need to be shared with high frequency that rerouting traffic to one instance will induce less overhead. Single instance strategy based on traffic rerouted through a single switch where a single instance of the security module would be deployed which would be decided by the VNF orchestration unit.

3.3.2.4 Ingress-control

Mitigation can be defined as a process that enables a victim server to continue serving requests in the presence of an attack. This process is usually temporary and can be uplifted once the attack subsides. For example, Somani et al. in [25] discuss mitigation and recovery methods for DDoS attacks in the cloud as resource scaling, victim migration, OS resource management, software-defined networking and finally DDoS mitigation as a service. Such mitigation methods are based on scaling resources or attack filtering which involves filtering the attack traffic and dropping it without disturbing legitimate traffic [25, 139]. Other methods are based on rate-limiting techniques where control the rate of traffic sent or received. As filtering or rate-limiting process include dropping or rate-limiting attack traffic

that consumes victim and infrastructure resources such as network bandwidth, Therefore, Ingress-control strategy is based on allocating functions to be close to traffic ingress point to reduce attack traffic entering the network, consume network resources, saturate network links and cause congestion.

3.3.3 Constraints

The placement of security VNFs, which are requested as security services by tenants over a virtualised environment in distributed locations collocated with network switches, casts as a resource allocation problem where different allocations for a security function will cost a different amount of computing resource and communication overhead of the infrastructure resources based on the allocation strategies proposed. We show this cost as traffic, resources and security constraints presented below.

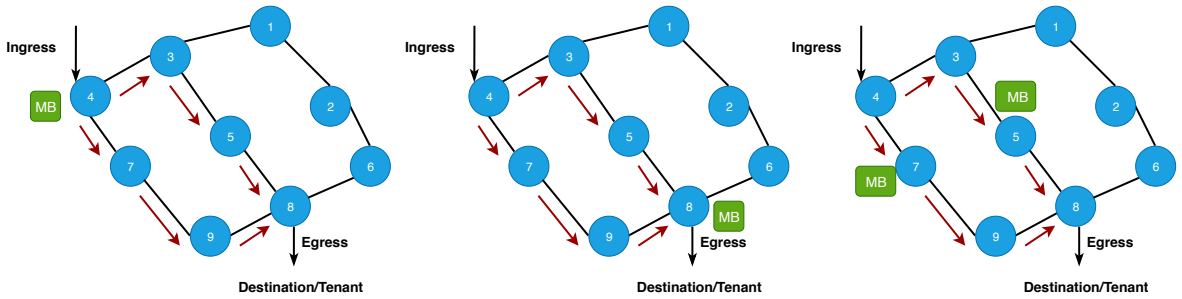
3.3.3.1 Traffic

The traffic constraints of a security module will limit the available locations where the required traffic granularity is satisfied. This will introduce computing and communication overhead based on the selected location. As shown for the following cases:

- The security modules of the stateless class process traffic on a per-packet or per-flow basis and, therefore, they can process traffic in parallel over a per-flow routing protocol. Consequently, they can adopt the independently duplicated strategy over links to cover the overall traffic distribution. For example in Figure 3.5, traffic enters the network through ingress point 4 and network switches/routers and links carrying the traffic for a tenant through egress point 8. Assuming flow-based ECMP routing where flows are distributed on equal cost paths and assuming that traffic from ingress 4 will be distributed on 2 equal costs shortest paths 3,5 and 7,9 to destination 8 as shown in red arrows. Therefore, a stateless security module requested for a tenant through egress

8 will have three allocations that satisfy the traffic constraints of the function to be considered:

- Ingress allocation: a single instances of the function is deployed at the ingress point, as shown in a green block in Figure 3.5a which represent minimum resources consumption and overhead presented by deploying only one instance and no redirecting of traffic is required.
- Egress allocation: a single instance of the function is deployed at egress switch covering all traffic destined to/originating from the tenant's host as shown in a green block in Figure 3.5b which represents minimum resources consumption as well.
- Intermediate allocation: multiple instances of the function are deployed at intermediate switches as long it covers all the traffic to/from the tenant. For example at points 5 and 7 as shown in Figure 3.5c which represent a computing resources overhead due to duplicating instances deployment but no redirecting of traffic is required.



(a) Ingress Allocation (b) Egress Allocation (c) Intermediate Allocation

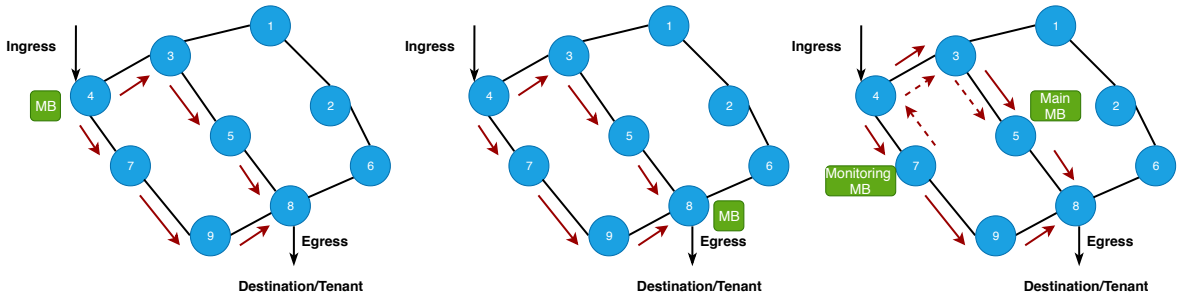
Figure 3.5: Traffic Constraint for Independent Duplication of the Stateless Class

- The security modules of the stateful class process traffic to extract anomalies based on coarser granularity than a single flow. Consequently, they adopt a dependent duplicated strategy over links, and a dependent instance will be deployed to cover the overall traffic distribution. Monitoring nodes will be deployed wherever traffic is distributed. These nodes will extract the needed features from the traffic and share this

information with the main node that takes the detection decision. For example, in Figure 3.6 traffic enters the network through one ingress point and out through one egress point to the tenant's host and routed the same as figure 3.5. A stateful security module requested for the tenant will have the same three allocations to the stateless shown in Figure 3.5 except in case of multiple instances deployment. One main instance at one point and monitoring instances for the rest of the points will be deployed, as shown in figure 3.6.

- Ingress allocation: a single instance of the function is deployed at the ingress point, as shown in a green block in Figure 3.6a which represents minimum resources consumption.
- Egress allocations: a single instance of the function is deployed at egress switch covering all traffic destined to/originating from the tenant's host as shown in a green block in Figure 3.6b which represents minimum resources consumption.
- Intermediate allocations: multiple instances of the function are deployed at intermediate switches as long it covers all the traffic to/from the tenant. One of them will be the main instance and the others will be monitoring instances of the function. For example, a main instance at points 5 and a monitoring instance at point 7, as shown in Figure 3.6c. Of course, the other way around with the main instance in 7 and the monitoring instance at 5 is valid too. This allocation requires a computing overhead due to duplication and a communication overhead due to coordinating communication between instances.

It is important to consider the communication that carries the metadata information between the monitoring node in 7 and the main node in 5 as shown in Figure 3.6c in dotted lines as it will impose a communication overhead to the allocation. Moreover, the path for that communication is a multi-path for instance, the communication path between 7 and 5 can follow the path 4,3 or 9,8. Minimising this overhead must be considered, such as selecting the path with minimum cost in case different paths are available with different costs.



(a) Ingress Allocation (b) Egress Allocation (c) Intermediate Allocation

Figure 3.6: Traffic Constraint for Dependent Duplication of the Stateful Class

- A single-instance allocation will be adopted when independent or dependent duplication is not accepted. In such case, all traffic must be routed to one instance of the function as shown in Figure 3.7. A security module requested for the tenant will have multiple allocations that satisfy the traffic constraints where every available point for allocation is to be considered in case of single instance allocation strategy. This strategy will impose restrictions on routing the traffic such as in the case of intermediate allocation as shown in Figure 3.7 where multiple paths are dropped.

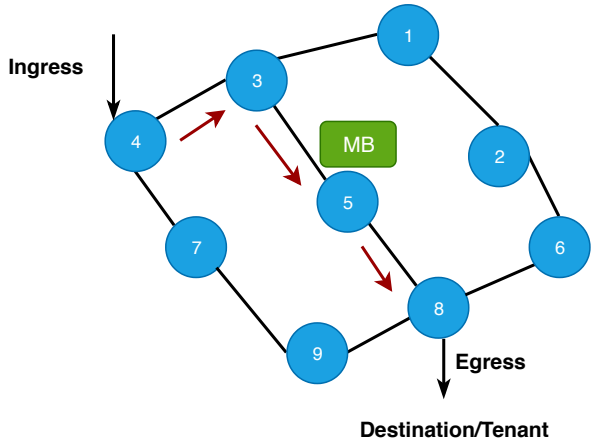


Figure 3.7: Traffic Constraint for Single Instance

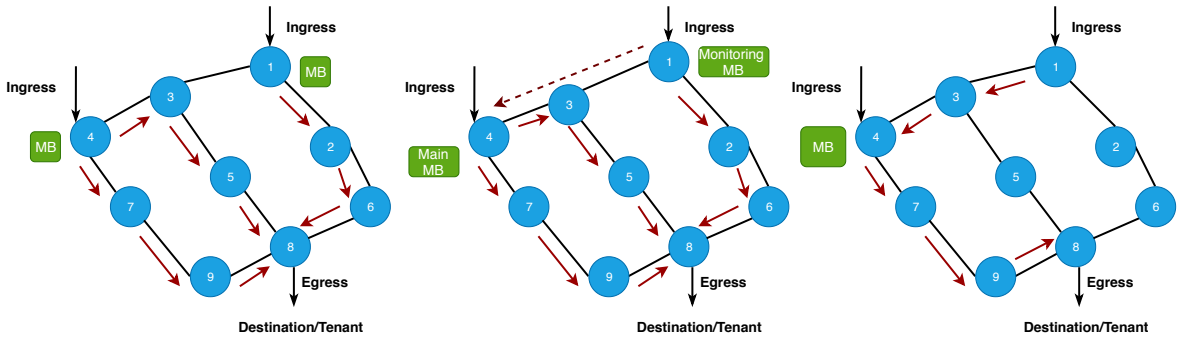
- In case of multiple ingress point network, ingress allocations will require multiple instances to be deployed at each ingress point as shown in Figure 3.8 and will be treated like intermediate allocation.

  – For a stateless class: duplicated instances of the function are deployed at each ingress point, as shown in a green block in Figure 3.8a which represent overhead

on resource consumption in the form of duplication.

– For a stateful class: multiple instances of the function is deployed at each ingress switch covering all traffic destined to/originating from the tenant's host one main and monitoring instances for the rest as shown in the green block in Figure 3.8b which represent computing resources overhead in the form of duplication and communication overhead in the form of communication between monitoring and main instances as shown as a dotted line.

– for single instance: single instances of the function are deployed at one ingress point and rest of traffic will be routed to that point as shown in Figure 3.8c with communication overhead in form rerouting traffic to a non-shortest path.



(a) Stateless Allocation (b) Stateful Allocation (c) Single Instance Allocation

Figure 3.8: Traffic Constraint for Multiple Ingress Networks

3.3.3.2 Resources

While each location has limited computing resources defined by a vector (CPU cores, Memory...etc.), each request from a tenant must be associated with a similar vector of the estimated resource required which is the resources required for the modules to process the required amount of traffic. The chosen allocation must then satisfy the resource requirements of the request where the resources available at the chosen location(s) must be greater than or equal to the resources requested, and only the allocations that satisfy the traffic and resource constraints are considered in the selection process. However, computing the resources required by a security module is a complex process. For any traffic processing application

such as a security function, the temporal traffic characteristics can have a significant impact. Moreover, beyond a certain load, any increase in the traffic rate will cause packets to be delayed and possibly dropped. For example, reports indicate that stateless IDSs such as Snort and Suricata start dropping/passing packets in case of dealing with a large amount of traffic, high speed or large packet size [140]. Thus, assigning resources to security modules must ensure that instances would process traffic at line rate with no packet drop.

We conclude that the resources required by a security module depend on many factors such as hardware, configuration, platform, the rule set, traffic rate, etc [140–145]. However, when estimating the resources for the module deployment, traffic intensity is the main factor to consider under the same platform. Thus, each security module available to be requested by tenants will be associated with a resources vector that represents i) *baseline resources* which is the amount of resources required for initial deployment of the module and ii) *traffic resources* which is resources per traffic unit associated with each traffic type such as (e.g TCP, HTTP, mixed traffic, etc.) and represents the estimated amount of resources required to process a unit of traffic of this type. Besides, each request will be associated with the estimated rate(s) of each traffic type(s) to be processed for the requested tenant. Similar approaches have been adopted by other researchers in allocating computing resources, in [146] the two parts of resources named ”rigid” and ”fluid”, while in [147], they were ”load-independent” and ”load-dependent”. From the service provider point of view, *baseline resources* and *traffic resources* for a module are to be determined empirically by testing each module in the modules pool over infrastructure [148].

In case of mitigation modules with a filtering approach that results in dropping a significant amount of traffic before saturating network links, an ingress-control strategy will be adopted. The strategy will restrict the allocation to be close to the ingress point to ensure minimum disturbance to tenant and infrastructure. It also allows saving network bandwidth to other tenants and reduces overall latency.

3.3.3.3 Security Constraints

A security-related constraint has been proposed by researchers that include forcing/preferring allocating certain services or services' requested by a specific tenant to certain locations for security reasons. These rules will be forced through security constraints in the placement and their cost will depend on which strategy allocation will be used.

3.3.4 Objective Function

The rest of this chapter will focus on the allocation of security modules over a virtualised environment infrastructure using a *resource-aware placement methodology*. We focus on security services placement in multi-tenant virtualised data centers where they are offered as software modules that are allocated throughout the infrastructure as VNFs to process the required traffic. The services are offered on a per-tenant basis as, and each request will result in deploying one or more instances of security functions to process traffic of the requesting tenant. The placement algorithm will be responsible for allocating the requested functions to locations that satisfy the traffic, resources and security constraints detailed in the previous section.

Since there will be more than one allocation that will satisfy the constraints of the request, the framework will select one that optimises the objective function. For efficient management of resources, we design the *two-dimensional* placement to optimise the resources usage by joint optimisation of communication and computing resources. We accomplish this with two objectives; The first objective is to maximise the *residual resources* of the framework, which represents the spare computing resources of the locations after the placement has been completed. The second objective is to minimise the *communication overhead* of the allocation which represents the communication bandwidth overhead due to the allocation and targeting minimise bandwidth usage for the placement.

3.4 Mathematical Model

In the rest of the thesis, we focus on the static placement problem of security modules in a virtualised multi-tenant environment (e.g. data centers) considering the computing and network resources. To satisfy the traffic constraints, the *stateless class* modules will independently duplicate over links in case of allocations where traffic is distributed over multiple links. In contrast, the *stateful class* modules will dependably be duplicated as one main instance and monitoring instances that will be duplicated over links, where traffic is distributed. To satisfy the resources constraint, only locations with computing and communication resources more than the resources required by the security module will be considered for allocation. The placement framework will be forcing the security constraints, restricting allocation to predefined location(s) and the single instance allocation too. The allocation that satisfies all the constraints will be considered in the placement, and the final selection for allocation for each requested module will be based on optimising the objective function proposed in Section 3.3.4. In this section, we model the placement problem as an instance of a bin packing problem.

3.4.1 Formulation

The placement of security functions is an instance of a variable cost – variable size bin packing problem (VSBPP) [149]. The bins represent switches and links each with capacity that represents computing resources and bandwidth respectively. The requested security modules are the items, bin sizes are the resource capacity of the switches and bandwidth capacity of the links. The cost of allocating a request to a location will have two dimensions, computing and a communication cost. To represent the problem as VSBPP, every request must be allocated to only one location. Therefore, every request will be associated with a tenant and a security function requested by that tenant.

A list of the available locations for the deployment will be defined and considered for the allocation. Every location available has two dimensions of resources, switches dimension

and links dimension. The switches dimension will represent the set of switches for a request to be allocated, while the links dimension will be the links representing the function communication overhead to allocate the request in the switches. Egress locations that represent allocating requests in egress switches can be used to allocate both types of security functions and have empty links dimension, while locations representing other allocations are designed for only one type. Therefore, every request can be only associated with locations that could allocate its function type (stateless, stateful and single-instance).

For example, in Figure 3.5a, the location of the stateless class, shown in green, will be represented as two dimensions location, switches dimension as shown in equations 3.1 and links dimension shown in equation 3.2. However, the links dimension will be empty as shown:

$Switches\_dimension$

$$Location.S=\{s_4 : main\_instance\} \tag{3.1}$$

and $links\_dimension$

$$Location.L=\{\} \tag{3.2}$$

Another example, in Figure 3.5c the location of the stateless class, shown in green, will be represented as shown in equations 3.3 and 3.4.

$Switches\_dimension$

$$Location.S=\{s_5 : main\_instance, s_7 : main\_instance\} \tag{3.3}$$

and $links\_dimension$

$$Location.L=\{\} \tag{3.4}$$

Another example, in Figure 3.6c the location of the stateful class, shown in green, will be represented as shown in equations 3.5 and 3.6:

$Switch\_dimension$

$$Location.S = \{s_5 : main\_instance, s_7 : monitoring\_instance\} \tag{3.5}$$

and $links\_dimension$

$$Location.L = \{[(s_7, s_4)], [(s_4, s_3), [(s_3, s_5)]\} \tag{3.6}$$

Locations will cover all allocation possibilities for requests, for example, there is another allocation where the main instance will be in s_7 and the monitoring instance will be in s_5 instead of the other way in the example. Every location will be associated with a request, but a request can be allocated to only one location. For example, considering the shortest path routing between monitoring and main instances, another location can be considered with the same switches dimension, but for the links dimension, it will use the communication path through switches 9 and 8.

The cost of different allocations is presented as the sum resources consumed of the allocation in both dimensions and will increase with duplicated/monitoring instances deployed in the architecture to cover traffic distribution. Therefore, to minimise this cost and optimise the placement to our objectives presented in 3.3.4, the allocation will reduce duplicates and subsequently keep allocation in minimum resource consumption locations while maintaining all constraints of allocating the functions satisfied. For simplicity, we assume switches resources is a one-dimensional vector.

The formulation of the problem is as follows: Let the overall framework include a set of $q > 0$ requests $Q = \{r_1, r_2, ...r_q\}$ with each request r representing a function requested by a tenant each, set of $m > 0$ switches $S = \{s_1, s_2, ..., s_m\}$ each with attribute $s.c$ as the resources capacity of s, a set of $n > 0$ links $L = \{l_1, l_2, ...l_n\}$ each with attribute $l.b$ as the bandwidth capacity of link l and attribute $l.w$ as the link weight, a set of k locations $P = \{p_1, p_2, ...p_k\}$, switches and links dimensions for locations for different requests presented in the following two metrics. A matrix u of size k x q x m representing the switches

resources cost of allocation where $u_{p,r,s}$ is location p required resources to be allocated in switch s to satisfy request r; or zero if location p can not accommodate request r due to function type or does not require any resources to be allocated in switch s. A matrix v of size k x q x n representing the communication cost of allocations where $v_{p,r,l}$ is location p required bandwidth to be allocated in link l to satisfy request r; or zero if location p can not accommodate request r due to function type or does not require any bandwidth to allocated in link l. A matrix w of size k x q represents the validation of allocating requests to locations where $w_{p,r}$ equal 1 if the location p valid for satisfy request r or zero otherwise. This matrix enforces security constraints when allocating security modules are constrained to specific servers, also it will be used to force allocation security class type match.

We have two objectives: the first objective is to maximise the *residual resources* (RS) which represents the spare resources in switches after placement. The second objective is to minimise *Communication Overhead* (CO) which is the communication cost of the placement that is represented as the sum of communication overhead traffic rate on each link multiplied by the link weight. The allocation is represented as a binary variable x of size qxk where $\mathbf{x}_{r,p}$=1 when request r is allocated to location p; 0 otherwise. The problem variable $\mathbf{x}$ is represented as

$$\mathbf{x}_{r,p} \in \begin{cases} 1, \\ 0 \end{cases} \qquad \forall r \in Q, \;\; \forall p \in P \tag{3.7}$$

The formulation is as follows: , i.e.:

$$\max . \Big[\sum_{\forall s \in S} s.c - \sum_{\forall r \in Q} \sum_{\forall p \in P} \sum_{\forall s \in S} \mathbf{x}_{r,p} \cdot u_{p,r,s} \Big] \tag{3.8}$$

$$\min . \Big[\sum_{\forall r \in Q} \sum_{\forall p \in P} \sum_{\forall l \in L} \mathbf{x}_{r,p} \cdot v_{p,r,l} \cdot l.w \Big] \tag{3.9}$$

$$\text{s.t.} \sum_{\forall r \in Q} \sum_{\forall p \in P} \mathbf{x}_{r,p} \cdot u_{p,r,s} \ \leq s.c \quad \forall s \in S \tag{3.10}$$

$$\sum_{\forall r \in Q} \sum_{\forall p \in P} \mathbf{x}_{r,p} \cdot v_{p,r,l} \ \leq l.b \quad \forall l \in L \tag{3.11}$$

$$\mathbf{x}_{r,p} = 0, \quad \forall r \in Q, \forall p \in P \qquad if \ w_{p,r} = 0 \tag{3.12}$$

$$\sum_{\forall p \in P} \mathbf{x}_{r,p} = 1, \quad \forall r \in Q \tag{3.13}$$

The first objective function of the formulation is represented in equation (3.8) as maximising the residual resources after the placement. The Second objective function of the formulation is represented in equation (3.9) as minimising the communication overhead after the placement. The constraint in equation (3.10) represents the switches capacity constraints. The constraint in equation (3.11) represents the links capacity constraints. The constraint in equation (3.12) represents the location-validity for requests where security constraints and the function type of the request must be satisfied by the location. The constraint in equation (3.13) ensures that each request is allocated to one location.

3.4.2 Security Placement Reduction to VSBPP Problem

As classic bin-packing problems have been shown to be NP-hard [150]. To prove that the placement of security modules in multi-tenant data centers is NP-hard problem, we reduce the known NP-hard problem variable sized bin packing problem (VSBPP) with variable cost to our placement problem in polynomial time. In VSBPP a set of items is going to be allocated to a set of bins. The bins have different sizes and different costs, and the objective is to minimise the overall cost of bins used for packing the items. Consider the security modules requests are the items, consider each switch and link be a bin with limited capacity.

Given an instance of the VSBPP, we can transform it to an instance of the placement problem in polynomial time.

- Create a list of locations that represent the combinations of switches and links for each possible location and which security class can be allocated to as described in Section 3.4.1.
- Allocate requests to locations with matching type.
- Extended the cost attribute of each item to represent the consumed size of each location.
- The objective still to minimise the cost of bins representing the amount of resources consumed.

The above reduction is trivial and can be carried in polynomial time. Therefore, the VSBPP problem is reducible to the security function problem in polynomial time, and hence the problem is NP-hard.

3.5 Solution Methods

Since the VSBPP is NP-hard, exact methods can not solve large instances of the problems within a reasonable time. While not-optimal/ approximated solutions are proposed to solve NP-hard problems, their accuracy varies. Some of the thesis algorithms have approximation guarantee accuracy (e.g. First Fit Decreasing), and thus they are not problem-specific, others like local search based algorithms have no accuracy guarantee and mainly depend on the shape of the search space of the problem [151]. Exploring the optimality of different algorithms for our placement problem would result in a significant reduction in resource usage in a virtualised environment such as cloud data centers.

Therefore, we explore various approaches used to solve similar problems such as VSBPP and the VNF placement problem to determine their optimality and performance in solving

the placement of security functions problem. For example, Heuristic algorithms are simple to design and implement and very efficient as well. However, it can be very different in effectiveness according to the different problems. Some are approximated or feasible solution guarantee while others can not guarantee a feasible solution by the end of the algorithm. Greedy Constructive heuristic is one of the most common used heuristic algorithms and many have proposed it to solve similar problem to our placement problem. It is usually used in cases of very little compute time and very large instances such as First-Fit Decreasing (FFD) and Best-Fit Decreasing (BFD) algorithms that have been proved to be approximated guarantee in solving classic bin-packing problems within a polynomial time [152]. Moreover, the most common meta-heuristics algorithms used in the literature are the Genetic Algorithm (GA), Greedy Randomized Adaptive Search Procedure (GRASP), Neural Networks (NN), Threshold Accepting Algorithms (TAA), Simulated Annealing (SA), and Tabu Search (TS). However, accuracy of the meta-heuristic algorithms is based on the search space of problem. We adopt a Tabu search based algorithm to solve the security function placement as it has been used in bin packing problem [153] and recently in VNF placement problems [154, 155] and it also showed better performance over other meta-heuristics algorithms for large size instance when solving bin packing instances [151]. In the following, we describe the adoption of these solutions to solve the security function placement problem:

3.5.1 Constraint Programming

The first approach is the constraint programming solution which gives an optimal solution to the problem. However, the problem has two objectives which requires combining the two objectives for the optimiser to solve. Sum the two objectives as stated in equation 3.14 is a valid method to address this problem with maximising the residual resources objective in equation 3.8 converted to minimise consumed resources. While other methods can be used such as two-pass optimisation where it optimises the problem to one of the objectives, then it optimises to the second objective. We target to find the optimisation that balances the two objectives.

$$\min . \Big[\sum_{\forall r \in Q} \sum_{\forall p \in P} \sum_{\forall l \in L} \mathbf{x}_{r,p} \cdot v_{p,r,l} \cdot l.w + \sum_{\forall r \in Q} \sum_{\forall p \in P} \sum_{\forall s \in S} \mathbf{x}_{r,p} \cdot u_{p,r,s} \Big] \tag{3.14}$$

3.5.2 Heuristic

Constructive heuristics algorithms start with 1) an empty subset/solution then 2) at each iteration it selects an element that is believed to be the best one or admissible one and will result in getting an optimal or feasible final solution. Then 3) insert the element into the current subset. 4)go to step 2 until a solution is complete. We detail some of the most common used greedy algorithms for the classic bin-packing problems as follow:

First-Fit Decreasing (FFD) and Best-Fit Decreasing (BFD) algorithms are decreasing algorithms have been proved to be approximated guarantee in solving classic bin-packing problems within a polynomial time, with an approximation of not more than $\frac{11}{9} \cdot N + 1$ bins, where N is the number of bins in an optimal solution [156]. There is also a simple heuristic such as First Fit which is the same as FFD without ordering elements in non-increasing order. In the following we explain the FFD algorithm:

1. Sort the bins arbitrarily
2. Sort the items in non-increasing order
3. Start with the empty solution
4. Select the first unassigned item in order
5. Select the first bin to receive the item (has enough space, choose empty bin if necessary)
6. Insert the new assignment in the solution
7. Go to step 4 until no item is unassigned

The BFD algorithm is most widely applied to bin-packing, VM placement and VNF placement problems. In BFD, the items are still sorted in decreasing order as in step 2 of FFD,

but the sorted items are allocated to the best location where the best location is the one with minimum space left after the allocation [157]. However, Best-fit can be interpreted in different contexts. For example, in [149] Beloglazov et al. solve a VM placement problem with the objective to save power consumption. They used a power-aware best fit decreasing algorithm that allocates virtual machines (VM) to locations that cause the least increase in power consumption as a best-fit location. Some approaches add location/bins order to the algorithm such as in [158] authors sort hosts with the highest current utilisation among the least increased power consumption to increase energy efficiency. Thus, we propose adopting a *resource-ware BFD* decreasing algorithm to solve the placement of security functions with resources usage optimisation. The decreasing order of the requests will be based on required resources, and the best-fit location will be selected based on minimum resources consumption to optimise our objective.

3.5.3 Meta-Heuristic

Tabu search is based on local search (search among neighbours). It is proposed to solve the local search problem of stuck on sub-optimal solutions by allowing non-improving moves when a local optimum is encountered. It searches feasible solutions to find one that optimises the objective function as well as local search. It starts by an initial solution then moves to the next solution by searching the surrounding neighbouring of the current solution for a solution better than the best feasible solution found by the algorithm so far. Then repeat the operation with the new obtained solution until no improvement can be found or time/iteration limit has been exceeding. However, tabu search broke the rule of moving to a better solution by moving to a worst solution if no improvement solution can be found (local minimum). Furthermore, it utilises a tabu list of solutions that is forbidden to visit. The tabu list consists of recently visited solutions and can include user-defined solutions. The recently visited solution is stored in a memory structure to increase the chances of visiting new solutions. Nonetheless, aspiration criteria can remove a solution from the tabu list and override it to allow for improving the solution in the tabu list to be considered for moving. Besides, the

length of the tabu list and stopping criteria determine the complexity of the tabu search. It should not be set too short or too long to avoid local optima or high complexity. Moreover, the iterations of stopping criteria should be set reasonably to get a near-optimal solution without complex calculations.

We design the main five components of Tabu algorithm that fit the placement of security functions as follow:

1. Initial solution: initial solutions are examined for BFD and random solutions.

2. Neighbourhood solutions: Ideally, each feasible solution that involves moving each virtual function from one location to the other could be a neighbour solution. However, this can lead to a big search space, so we define the following neighbourhoods suitable for our security placement.

    - restrict the move to one function, with the largest resource size. If the function does not have any feasible neighbours, no location can accommodate this function, the next function in size is to be selected.
    - move can be shift or swap

3. Tabu list: Tabu list can be fixed size FIFO queue or memory structure. A memory structure is where a function is moved from one location to another then we add the move of this function go back to its original location to the tabu list , so it can not be moved for the next n-1 movies where n is the number of functions which give the chance to other functions to move before moving this function back. We implement memory type with size n.

4. Aspiration rule:

    - If a tabu move has a better solution than the most known solution.
    - If no moves available but the tabu moves, due to no moves with better solutions. Then we select the best tabu, best solution in the tabu list.

5. Stopping condition:

    - after m iteration.
    - no feasible solutions in the neighbourhood can be found.

3.5.4 Near-Optimal Subset-Sum Solution

We adopted a Subset-sum solution that has been used as near-optimal solutions in problems such as VM allocation [158] and variable size bin packing problems [159]. It is based on the constructive heuristic that requires to solve the subset-sum/knapsack problem, which is solved optimally in pseudo-polynomial time. It includes finding the best-fit functions for each location, instead of finding the best location for each function. We can solve this problem as a 0-1 knapsack problem for each location.

3.5.5 One-dimensional Implementation

Implementation of the placement problem can consider the computing resources and eliminates the communication overhead, which converts the problem to a one-dimensional resources allocation problem. To eliminate the communication overhead and be able to satisfy the traffic constraint, the stateful class modules will be allocated to switches where they can capture all traffic destined to the requested tenant such as single ingress allocation or egress allocation. Of course, this implementation is limited to architectures where at least one allocation available for every user where are all terrific is passing through one point the same as in egress allocation in Section 3.3. For-short, we reference the two implementations as a one-dimensional model and two-dimensional model.

The same as the two-dimensional model, the one-dimensional model is optimising the placement problem to achieve efficient management of resources. Therefore, the objective is to maximise the *residual resources* of the framework which represent the spare resources after the placement has been completed. While the allocation resources cost is represented by the sum of the computing resources consumption of VNFs allocated to satisfy tenants requests

and will be increased with duplicated instances of modules deployed in higher layers of the hierarchy of the architecture to cover traffic distribution.

The formulation of problem is as follows: Let the overall framework include a set of $q > 0$ requests Q= $\{r_1, r_2, ...r_q\}$ with each request r representing a function requested by a tenant, a set of $k > 0$ locations P= $\{p_1, p_2, ...p_k\}$, each with attribute $p.c$ as the resources' capacity at location p which represent the sum of switches capacity in the location. A matrix w of size k x q representing the validation of allocating request to locations where $w_{p,r}$ equal 1 if the location p valid for satisfy request r or zero otherwise. For instance, for the stateful class, only the ToR level will be valid for allocation. While, for the stateless class, there are three parent locations that are valid. A matrix u of size k x q representing the locations resource cost of allocation where $u_{p,r}$ is the required resources to be allocated in location p to satisfy request r; or zero if location p can not accommodate request r due to function type or does not require any resources to be allocated. The allocation is represented as a binary variable $\mathbf{x}$ of size qxk where $\mathbf{x}_{r,p}$=1 when request r is allocated to location p; 0 otherwise. The problem variable $\mathbf{x}$ is represented as

$$\mathbf{x}_{r,p} \in \begin{cases} 1, \\ \\ 0 \end{cases} \qquad \forall r \in Q, \;\; \forall p \in P \tag{3.15}$$

The formulation is as follows: , i.e.:

$$\max . \Big[\sum_{\forall p \in P} p.c - \sum_{\forall r \in Q} \sum_{\forall p \in P} \mathbf{x}_{r,p} \cdot u_{p,r} \Big] \tag{3.16}$$

$$\text{s.t.} \sum_{\forall r \in Q} \mathbf{x}_{r,p} \cdot u_{p,r} \;\; \leq p.c \quad \forall p \in P \tag{3.17}$$

$$\mathbf{x}_{r,p} = 0, \quad \forall r \in Q, \; \forall p \in P \qquad if \; w_{p,r} = 0 \tag{3.18}$$

$$\sum_{\forall p \in P} \mathbf{x}_{r,p} = 1, \quad \forall r \in Q \tag{3.19}$$

The objective function of the one-dimensional formulation is represented in (3.16) as maximising the residual resources after the placement. The constraint in (3.17) represents the location's capacity constraints which are enforced by the resources requirements of the problem. The constraint in (3.18) represents the location-validity for requests which enforce the traffic constraint of the problem and the stateful allocation constraint. The constraint in (3.19) ensures that each request is allocated to only one location.

While the one-dimensional consider computing resources of the architecture, previous methods can be adopted to solve the implementation by adding the placement of stateful class modules in single switch locations as a constraint.

3.6 Summary

In this chapter, The framework proposed has used the SDN and VNF technology to address the limitation of legacy deployment of network function. It has tackled the specific constraints of security modules that previous virtualised network management systems failed to address. It has exploited features like on-path deployment and non-shared models to save resources and reduce management complexity. The framework allocation strategies are based on the classification of security modules that depends on the granularity of traffic processing in the module. These strategies have combined the efficient management of network resources and satisfying the traffic granularity required by the security module.

The resource-aware placement is responsible for selecting the final allocation of the security functions requests by satisfying the constraints of the placement presented by traffic, resources and security constraints and maintaining efficient management of resources by optimising the placement based on optimising the computing and communication resources objectives. Furthermore, we propose a one-dimensional implementation to the placement

problem that will eliminate the communication cost of the problem; however, it will restrict the stateful class module to locations where all traffic flows are passing through.

Chapter 4

Implementation

4.1 Overview

The placement of security modules is a resource allocation problem, where modules requested by tenants in virtualised data centers are allocated in a set of distributed locations each with limited capacity. We focus on the initial placement of security equivalence classes introduced in Chapter3. The following sections present the implementation of the proposed framework on fat-tree as the most common virtualised data centers architecture. Moreover, the implementation of the allocation strategy and the placement solutions on fat-tree will be demonstrated.

The fat-tree architecture and its characteristics are presented in Section 4.2. Then the implementation of the allocation strategies will be discussed. Then, we represent the constraint programming solution of the placement problem in Section 4.3. In Section 4.4, the implementation of the heuristic solution algorithm will be demonstrated. In Section 4.5, the implementation of a Meta-heuristic Tabu algorithm is presented. Then, a subset-sum Near-optimal solution is discussed in Section 4.6. In Section 4.7, the LP solution of the one-dimensional model of the placement problem is introduced. Finally, we conclude the chapter in Section 4.8.

4.2 Architecture

4.2.1 Fat-Tree Data Centers

A three-tier architecture is one of the common network architectures and considered the most common in virtualised data centers [160]. It consists of three structural layers. An access tier, the bottom level, in which each server is connected to one (or two for redundancy) access switch. Each access switch connected to one or two switches at the aggregation tier. An aggregation tier, in which each aggregation switch connects with multiple switches at the core tier. Many data center architectures follow the three-tier architecture to a great extent such as VL2, Fat-tree(Clos) and Bcube with every one of them has been built with a specific purpose in mind, for example, VL2 is an agile architecture that targeted load balancing, flat addressing and performance isolation between services [5].

The fat-tree topology, also known as Clos or Portland topology, is a scalable and fault-tolerant architecture which is built around the concept of pods [161]. A pod is a group of access and aggregation switches that form a Clos graph. Each pod is connected to the core switches with another Clos, by evenly distributing the up-links between all the aggregation switches of the pod and the core switches. For a fat-tree with the size k, The core layer contains $k^2/4$ k-port switches at the top. The aggregation layer has k pods, each containing two layers of $k/2$ switches. In the core layer, a port i in a core switch is connected to pod i. On the other hand, every switch in the access layer is connected to $k/2$ servers. Thus, a fat-tree with size k connects $k^3/4$ servers. For example, a k=4 fat-tree topology shown in Figure 4.1 with the three layers of switches (ToR, aggregation, and core). Fat-tree has a diameter of 6 regardless of its size. The fat-tree can build with commodity Ethernet switches, and the flows between 2 servers have multiple paths to consider. The core layer is the gate to the data center; it is responsible for all flows going in and out of the data center.

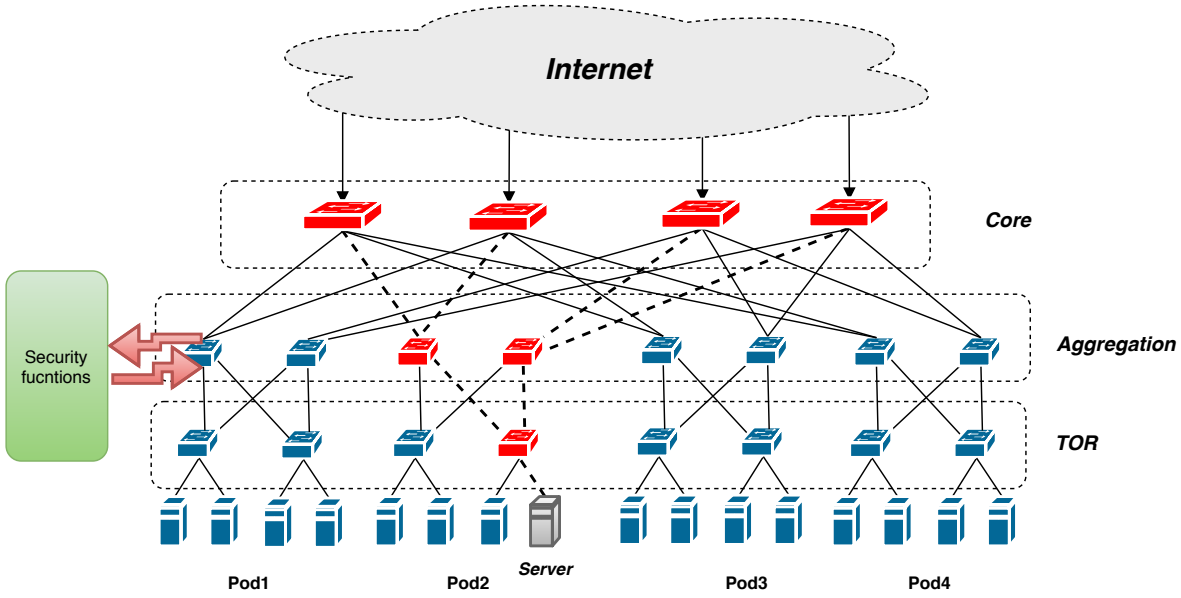


Figure 4.1: Fat-Tree Cloud Data Center Size k=4

4.2.2 Routing

We assume routing is flow-based Equal Cost Multiple Path (ECMP) [162] where a flow is identified by the typical 5-tuple (source IP, destination IP, source port, destination port, and the transport protocol type). In flow-based ECMP, flows are distributed over equal cost links which guarantee load balancing and in-order packet delivery within the same flow. For example, ”East-West” traffic destined to the server in grey in Figure 4.1 and coming from other pods will be distributed in the upward path until it reaches a core switch, and then it will traverse the dotted links in its descending path to the server. While ”North-South” traffic coming from outside the data center will follow a deterministic shortest path route through the dotted link to the destination.

4.2.3 Deployment Locations

Deployment locations for the security VNFs are collocated points to the network switches at all layers. Traffic is rerouted from switches to the security function and back. The security function abstraction can be implemented as switch/router-integrated computing modules or on a separate, virtualised commodity x86 architecture that physically connects to

a traffic-forwarding switch/router [163]. For example, a TCP acceleration network service implemented using HP's ONE (Open Network Ecosystem) server collocated with a network switch in [164]. The location selected to deploy a security function must satisfy the constraints imposed by the traffic processing methodology of the network function. We exploit the SDN enabled architecture to manage traffic in security functions and out to the destination.

For the capacity of the locations, design ensures that locations have enough resources to accommodate requests from tenants. To distribute allocation workload on all network switches, the designed architecture will distribute allocation workload (resources) equally on the three layers in case of a heavy load of requested resources. Because duplication will increase allocation cost in higher levels, switches initial capacity will increase for higher layers based on duplication cost which depends on network size. Therefore, we assign switches initial capacity in each level based on network size. For simplicity, we assume an equal workload of both classes. If x the *server_request_capacity* is the maximum requested resources for each server and that baseline and traffic part is 50% each, and as there is no duplication on TOR level, a TOR switch location will have an initial capacity of 33% of $server\_request\_capacity * number\_of\_servers\_per\_switch$ as shown in equation 4.1.

$$TOR\_capacity = 0.33 \cdot x \cdot k/2 \tag{4.1}$$

For aggregation level, there is the same number of switches as in TOR level and to accommodate the same amount of resources accommodated to TOR level, for each switch location duplication will increase the total resources requested by $(k/2 + 1)/2$ for the stateless class and by $1 + ((k/2 - 1)/(k/2)) \cdot 1/2$ for the stateful class which expressed in equation 4.2.

$$Agg\_capacity = (((k/2 + 1)/2) \cdot 1/2 + (1 + ((k/2 - 1)/(k/2)) \cdot 1/2) \cdot 1/2) \cdot TOR\_capacity \tag{4.2}$$

For core level, to accommodate the same amount of resources accommodated to TOR level with half the number of switches in TOR level, there be $((k/2)^2 + 1)/2$ duplication for the

stateless class and $1 + ((k/2 - 1)/(k/2)) \cdot 1/2$ for the stateful class which expressed in equation 4.3.

$$Core\_capacity = (((k/2)^2 + 1)/2 + (1 + ((k/2 - 1)/(k/2)) \cdot 1/2)) \cdot TOR\_capacity \quad (4.3)$$

4.2.4 Allocation Strategy Implementation

The placement of security VNFs casts as a resource allocation problem where different allocations for a security function will cost a different amount of computing resource and communication overhead that are represented as traffic, resources and security constraints. The implementation of these constraints in the fat-tree is based on the three-tier architecture and the ECMP routing of the north-south traffic from the core to TOR servers. The resources constraints will be typically enforced as capacity constraint where valid allocation will be considered if there are enough resources to accommodate requests. In contrast, security constraints will be enforced as validation constraints where only valid locations are permitted for certain requests. For the Ingress-control classes and as we adopt a resource-aware approach to the problem, it will be constrained to level 2 core layer allocation to prevent the harmful traffic from entering the network and this will be enforced through validation constraints. Besides, the traffic constraint for the stateless class and stateful classes is as follow:

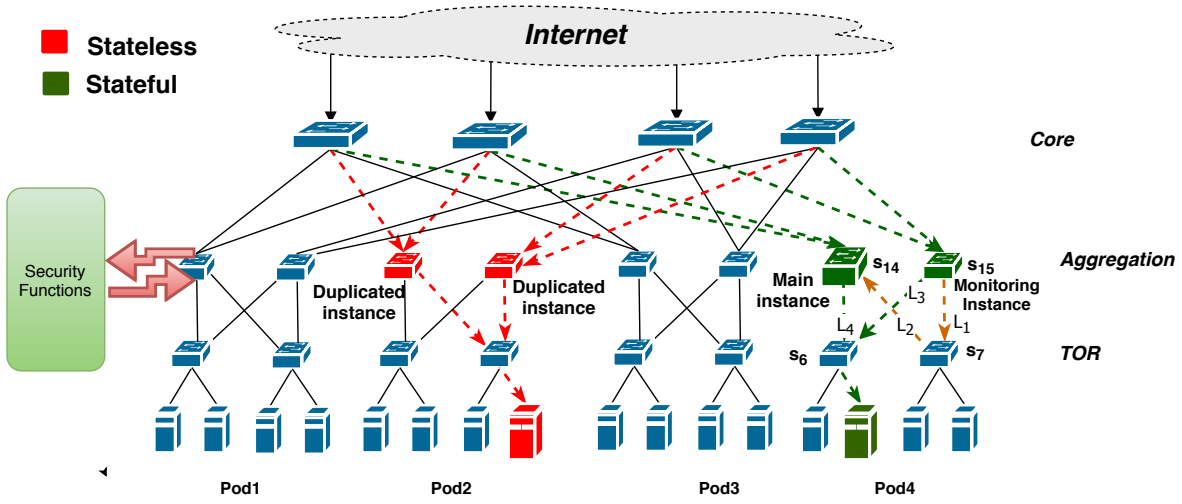


Figure 4.2: Traffic Constraint For the Stateless and Stateful Classes in Fat-Tree

4.2.4.1 Traffic Constraints for the Stateless Class

The security modules of the stateless class process traffic on a per-packet or per-flow basis and, therefore, they can process traffic in parallel over a per-flow routing protocol. Consequently, they can be independently duplicated over links to cover the overall traffic distribution.

In the fat-tree k=4 in Figure 4.2, switches and links carrying the traffic for the host in red are shown in dashed lines, and a security module requested for tenants residing on this host will have three allocations that satisfy the traffic constraints to be considered:

- Level 1 allocation: a single instance of the security function is deployed at the ToR switch of this server, covering all traffic destined to/originating from that host
- Level 2 allocation: two instances of the function are deployed at the aggregation switches routing traffic to/from this host, as shown in red in Figure 4.2.
- Level 3 allocation: four instances of the function are deployed in the four core layer switches.

4.2.4.2 Traffic Constraints for the Stateful Class

The security modules of the stateful class process traffic to extract anomalies based on coarser granularity than a single flow. Consequently, they can not be independently duplicated over links and dependent instances will be deployed to cover the overall traffic distribution. Monitoring nodes will be deployed wherever traffic is distributed. These nodes will extract the needed features from the traffic and share this information with the main node that takes the detection decision. In the fat-tree k=4 in Figure 4.2, a stateful security module requested for tenants residing on the host in green will have seven allocations that satisfy the traffic constraints to be considered:

- Level 1 allocation: a single instance of the security function is deployed at the ToR switch of this server, covering all traffic destined to/originating from that host

- Level 2 allocations: two allocations are available at this level, one allocation is where a main instance is deployed in s_{14} and a monitoring instance in s_{15} as shown in Figure 4.2 in green. The other allocation is the other way around with the main instance in s_{15} and a monitoring instance in s_{14}.

- Level 3 allocations: four allocations are available at this level. Each allocation will deploy only one main instance in one of the core switches and a monitoring instance in the other three core switches.

For stateful class functions, there is a communication overhead due to the dependent duplication allocation strategy. For example, the allocation shown in Figure 4.2, a communication that carries the metadata information between the monitoring node in s_{15} and the main node in s_{14} will impose a communication cost to the allocation. Moreover, The path for that communication is a multi-path for instance , the communication path between s_{15} and s_{14} can follow the path L_1, L_2 or L_3, L_4. This communication overhead requires the allocation process to consider the capacity of links. Furthermore, minimising this overhead must be considered, such as selecting the path with minimum cost in case different paths are available with different costs.

In cases where dependent duplication is not accepted, a single-instance allocation will be adopted, and all traffic must be routed to the allocation switch, as shown in Figure 4.3. To keep minimising the overhead, we restrict the locations available to the parents' switches of the host server. Therefore, a stateful security module requested for tenants residing on h_8 will have seven allocations that satisfy the traffic constraints to be considered in case of single instance allocation strategy.

4.3 Constraint Programming

The constraint programming model is combining the objectives of the mathematical model of the placement problem presented in Section 3.4. Two approaches have been considered

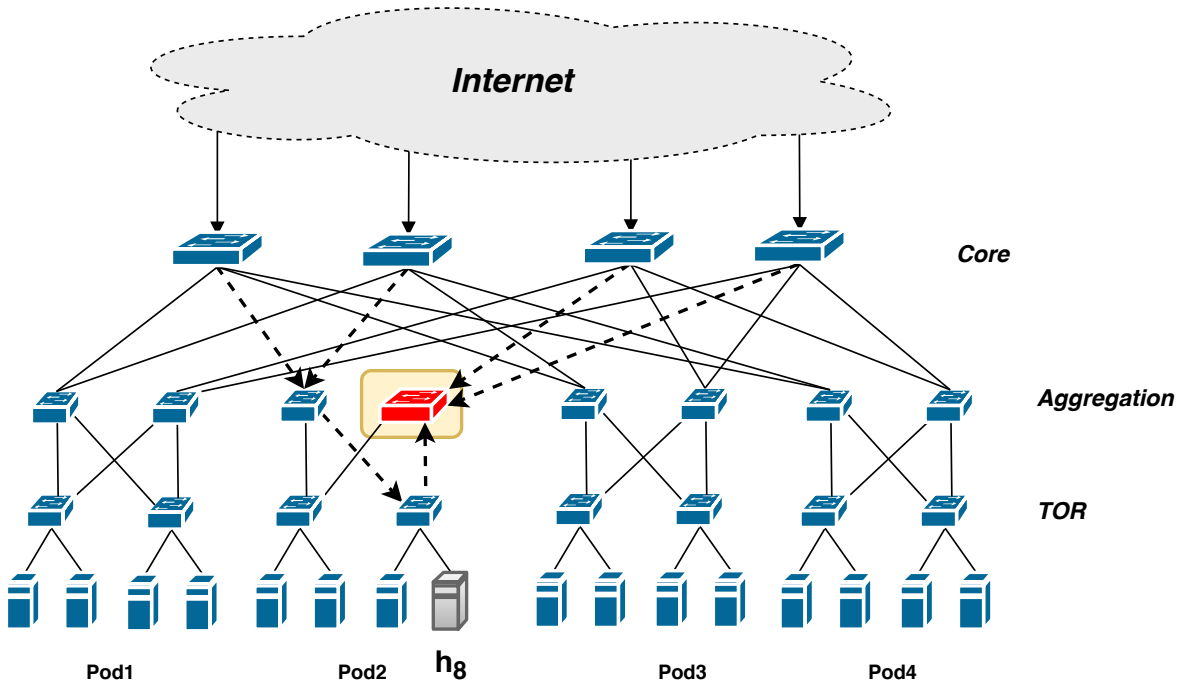


Figure 4.3: Traffic Constraint for Single Instance in Fat-Tree

to combine them. The first approach is called CP_COMP+COMM which optimises the constraint programming model for the sum of both functions with equal weights for each as stated in equation 3.14. This method gives both functions equal weight in the optimisation.

The second approach is called CP Two-pass [165], which consists of a two-pass method optimisation. First, it optimises the placement of functions of one class against one of the objectives then it optimises the functions of the second class for the other objective. In CP_2_PASS_STATEFUL version, the model optimises the placement of the stateful requests for minimising the communication overhead objective in equation 3.9 and then it optimises for maximising the residual resources objective in equation 3.8 against the stateless requests after encoding the solution obtained in the first optimisation as a hard constraint. This version optimises the objective in equation 3.9 as a primary goal to the problem, while the objective in equation 3.8 is considered a secondary goal. While the CP_2_PASS_STATELESS version optimises the stateless requests against the objective in equation 3.8 then the stateful requests against the objective in equation 3.9. The constraint programming solution implementation is shown in Algorithm 1.

Algorithm 1 CP Model Algorithm

Input: Set of requests Q, set of locations P
Output: Set of requests allocated to locations A

1: **function** GETCPSOLUTION(obj, Q)
2: $cpmodel \leftarrow CreatecpModel()$ ▷ Create model
3: $\mathbf{x} \leftarrow AddTwoDimBinaryVariable(cpmodel, Q, P)$ ▷ problem variable
4: **for all** $r \in Q$ **do**
5: **for all** $p \in P$ **do**
6: $AddValidConstraint(cpmodel, w(x,p), X(r,p))$ ▷ valid constraint
7: **for all** $r \in Q$ **do**
8: $AddLocationConstraint(cpmodel, r)$ ▷ single location per request constraint
9: **for all** $s \in S$ **do**
10: $AddSwitchCapacityConstraint(cpmodel, s)$ ▷ switch capacity constraint
11: **for all** $l \in L$ **do**
12: $AddLinkCapacityConstraint(cpmodel, l)$ ▷ link capacity constraint
13: $comp\_obj \leftarrow CreateCompObj(S, u, Q, P)$
14: $residual\_obj \leftarrow CreateResidualObj(comp\_obj)$
15: $comm\_obj \leftarrow CreateCommObj(L, v, Q, P)$
16: $comp\_plus\_comm\_obj \leftarrow CreateSumObj(comp\_obj, comm\_obj)$
17: **if** $obj{=}COMP$ **then**
18: $AddMaximizeObjective(cpmodel, residual\_obj)$
19: **else if** $obj{=}COMM$ **then**
20: $AddMinimizeObjective(cpmodel, comm\_obj)$
21: **else if** $obj{=}COMP\_PLUS\_COMM$ **then**
22: $AddMinimizeObjective(cpmodel, comp\_plus\_comm\_obj)$
23: **else if** $obj{=}TWO\_PASS\_STATELESS$ **then**
24: $Q^* \leftarrow GetStatelessRequests(Q)$
25: $stateless\_solution = GETCPSOLUTION(COMP, Q^*)$
26: $AddSolutionToModel(stateless\_solution, cpmodel)$
27: $AddMinimizeObjective(cpmodel, comm\_obj)$
28: **else if** $obj{=}TWO\_PASS\_STATEFUL$ **then**
29: $Q^* \leftarrow GetStatefulRequests(Q)$
30: $stateful\_solution = GETCPSOLUTION(COMM, Q^*)$
31: $AddSolutionToModel(stateful\_solution, cpmodel)$
32: $AddMaximizeObjective(cpmodel, residual\_obj)$
33: $solution = SolveCPModel(cpmodel)$ ▷ Solve model
34: $A \leftarrow GetAllocation(solution)$
35: **return** Set of allocated requests A

4.4 Heuristic Solutions

We adopt the resource-aware BFD algorithm introduced in Section 4.4 to solve the placement of the security functions problem. In the resource-aware BFD in Algorithm 2, Q is a set of initial requests, each represents a tenant's request to a certain module, P is a set of locations available for allocating requests and the set A refers to the set of allocated requests in certain locations, the function $sort(Q)$ sorts the requests from Q by a decreasing order of computing resources required; $capacity(A, r, p)$ ensures the resources required in location p in allocation A is enough to accommodate a give request r; $validation(r, p)$ constrains the location to those who satisfy the traffic constraints such as for a stateful class request r, only locations that associated with communication links are valid, while $total\_cost(r, p)$ calculates the total cost of allocating the request r to location p which represented of computing cost plus communication cost as illustrated by the following algorithm:

Algorithm 2 BFD Placement

Input: Set of requests Q, set of locations P
Output: Set of requests allocated to locations A

1: $A \leftarrow \emptyset$ ▷ initialisation
2: $Q^* \leftarrow sort(Q)$ ▷ sort request w.r.t. resources
3: **for all** $r \in Q^*$ **do**
4: **for all** $p \in P$ **do**
5: **if** ($capacity(A, r, p)$=TRUE) $\bigwedge$($validation(r, p)$=TRUE) **then**
6: $p^* = \arg\min_{p' \in P} total\_cost(r, p')$
7: **if** ($p^* \neq 0$) **then**
8: $A \leftarrow A \cup \{(r, p^*)\}$ // allocate request r to location p^*
9: **return** Set of allocated requests A

The proposed resource-aware allocation algorithm (i) sorts requests and then (ii) allocates each request to the *BF* allocation. Firstly, it sorts requests by the amount of required resources to deploy one instance for each in decreasing order. Then to determine the BF location, the cost of locating the request to each valid location will be calculated. Then, The algorithm selects the allocations that have enough resources to accommodate the request to satisfy the resources constraints, and finally selects the location that causes the least increase in total resource consumption (min cost).

For the fat-tree architecture, the cost is based on the architecture level and all locations on the same level have equal cost. Therefore, the algorithm is changed to the BFD in Algorithm 3 where $LEVELS\_LIST = [0, 1, 2]$ and $GetLocations()$ function return locations in certain level. The algorithm searches the architecture' levels in order from TOR to Core to find a location to the current request. If a location has enough resources to accommodate the request and satisfies the validation constraint, the algorithm stops the searching and assigns the location to the request. Although locations in the same level have the same computing and communication cost, the stateful class has an unequal distribution of computing resources on the switches of the location in aggregation and core layer where one switch will receive the main instance of the function. In contrast, other switches will receive a monitoring instance. To balance the load between switches in the same level and reduce fragmentation, a Worst-fit allocation will be adopted among the single level locations where locations with the most resources available will be allocated to requests first. This can be achieved by sorting location in non-increasing order of available resources as shown in step 6.

Algorithm 3 BFD Placement for Fat-tree

Input: Set of requests Q, set of locations P
Output: Set of requests allocated to locations A

1: $A \leftarrow \emptyset$ ▷ initialisation
2: $Q^* \leftarrow Sort(Q)$ ▷ sort request w.r.t. resources
3: **for all** $r \in Q^*$ **do**
4: **for all** $level \in levels\_list$ **do**
5: $P' \leftarrow GetLocations(level)$
6: $P* \leftarrow Sort(P')$ ▷ sort locations w.r.t. available resources
7: **for all** $p \in P*$ **do**
8: **if** $(capacity(A, r, p)$=TRUE$) \bigwedge (validation(r, p)$=TRUE$)$ **then**
9: $p^* = p$
10: **break**
11: **if** $(p^* \neq 0)$ **then**
12: $A \leftarrow A \cup \{(r, p^*)\}$ ▷ allocate request r to location p^*
13: **break**
14: **return** Set of allocated requests A

The non-increasing sort in step 3 is considering the computing resources (baseline_part+traffic_part) of the request, and we implement three versions of BFD algorithm with different biases. The tested BFD algorithms are BFD, BFD_STATELESS and BFD_STATEFUL. The BFD al-

gorithm has no bias, it sort requests in non-increasing order based on computing resources required for each while BFD_STATELESS is biased to stateless class functions where it sort requests of this class first then the stateful class, inside the same class it sort based on computing resources. The BFD_STATEFUL is the same as BFD_STATELESS but to the stateful class requests.

The heuristic algorithms RANDOM and FFD have been implemented. The RANDOM algorithm allocates requests in no specific order and selects locations randomly in a condition that the location has enough resources and is valid for the request. The FFD algorithm is the same as RANDOM but with requests sorted in non-increasing order before allocated. For N requests, the complexity of BFD and FFD algorithms for a fat-tree is $O(NlogN)$ for sorting the requests.

4.5 Meta-Heuristic Solutions

The Tabu search based algorithm has been used to solve similar problems to the security function placement and has shown better results in solving bin packing problems as previously discussed in Section 3.5. Therefore, we compare the greedy algorithm BFD to the TABU search method which starts by an initial solution then moves to the next solution by searching the surrounding neighbouring solutions of the current solution for a better solution than the current solution. We design the algorithm to optimise the objectives presented in Section 3.4. Thus, a neighbouring solution is considered better if it improves one of the objectives or both but never to worsen any of them. Typically, a move will consider being every possible move in the system; for example, in the security placement problem, a move could consider moving every request to every possible location. However, this will lead to very big search space, so the design will consider the search space consisting of every move that could be done by the most resources consuming request, if no better solution can be found, the next request in order is considered.

In the fat-tree architecture, multiple locations can fit the request, each with a different cost

where lower levels location has less cost than higher layers, so we implemented a *lower* move which includes moving the request down in the hierarchy to reduce resources. Furthermore, we consider a *swap* move which considers swapping the request location with every other request valid for the swapping process. The tabu_list will contain the recent requests that have been moved with a size equal to the number of requests. We implement three variants of the TABU algorithm with different moves types. The algorithms are TABU_LOWER, TABU_SWAP, and TABU_LOWER+SWAP where the last one is an algorithm that considers both movies (Lower and SWAP) for each request and selects the best of them to be the next move. The initial solution of the Tabu algorithms are tested with two initial heuristic solutions the BFD and RANDOM algorithms.

In the TABU_LOWER shown in Algorithm 4, Q is a set of initial requests, each represents a tenant's request to a certain module, P is a set of locations available for allocating requests and the set A refers to the set of allocated requests in certain locations, the function $sort(Q)$ sorts the requests from Q by a decreasing order of computing resources required; $InitialSolution()$ return initial solution to the problem; $GetBestMove(r)$ return a new location to request r that will most reduce computing or communication cost or both; $GetBestTabuMove(tabu\_list)$ will find the best move among all requests in $tabu\_list$; $CompareMoves()$ will compare to moves and return the one with most saving resources. $RemoveFirstElement(tabu\_list)$ will remove the first element of the $tabu\_list$ to implement the list FIFO queue.

For the TABU_SWAP, Algorithm 4 will be modified based on the swap move where $move$ will be a tuple of two moves indicating the swap and the $GetBestMove$, $UpdateAllocation$ and $CompareMoves$ function will be updated accordingly. The same for TABU_LOWER+SWAP algorithm where $move$ is the better solution of the lower or swap move. For N requests and M locations and Tabu list size L, the complexity of TABU_LOWER algorithm in worst case for a fat-tree is $O(NlogN+NL)$ where $O(NlogN)$ is the complexity of sorting requests and $O(NL)$ for best and tabu list moves. While the complexity of TABU_SWAP and TABU_-LOWER+SWAP, is $O(NlogN + NL + N^2)$.

Algorithm 4 TABU_LOWER algorithm

Input: Set of requests Q, set of locations P
Output: Set of requests allocated to locations A

```
1: A ← InitialSolution()                                   ▷ initialisation
2: Q* ← Sort(Q)
3: i ← 0
4: tabu_list ← ∅
5: while i ≤ MAX_ITERATIONS do
6:     for all r ∈ Q* do
7:         move = 0
8:         if r ∈ tabu_list then
9:             continue
10:        move ← GetBestMove(A, P, r)
11:        if move ≠ 0 then
12:            tabu_list ← tabu_list ∪ r                   ▷ Add r to the Tabu list
13:            A = UpdateAllocation(A, move)
14:            break
15:    if move=0 then
16:        move ← GetBestTabuMove(A, P, tabu_list)
17:        if move ≠ 0 then
18:            A = UpdateAllocation(A, move)
19:    if move=0 then
20:        break                                           ▷ no more moves- End algorithm
21:    if i ≥ TABU_LIST_SIZE then
22:        RemoveFirstElement(tabu_list)
23:    i ← i + 1
24: return Set of allocated requests A
25: function GETBESTTABUMOVE(A, P, tabu_list)
26:    best_move ← 0
27:    for all r ∈ tabu_list do
28:        move ← GetBestMove(A, P, r)
29:        if move ≠ 0 then
30:            if best_move=0 then
31:                best_move ← (r, p)
32:            else
33:                best_move ← CompareMoves(best_move, move)
34:    return best_move
35: function UPDATEALLOCATION(A, move)
36:    (r, p) ← move
37:    A ← A − (r, ∗)
38:    A ← A ∪ (r, p)
39:    return Set of allocated requests A
```

4.6 Subset-Sum Near-Optimal Solution

We adopted a Subset-sum solution that has been used as near-optimal solutions in problems such as VM allocation [158] and variable size bin packing problems [159]. It is based on a constructive heuristic that requires to solve subset-sum/knapsack problem, which is solved optimally in pseudo-polynomial time. It includes finding the best-fit functions for each location, instead of finding the best location for each function. Finding the best-fit functions can be solved as a 0-1 knapsack problem for each location.

The fat-tree will be rearranged to a new location schema to combine switches with duplicated instances in aggregation and core level to one location. The number of locations of the new schema in a fat-tree architecture will be $k^2/2 + k + 1$. $k^2/2$ locations in TOR level, one for each TOR switch. k locations in aggregation level, one for each pod and only one location to represent the core level locations. For instance, in the k=4 fat-tree, the number of new locations will be 13 as the blocks shown in Figure 4.4. Every new_location will invoke an instance of 0-1 knapsack problem that will consider all the valid and unallocated yet requests and all locations from the old location schema that require allocating resources in the new-location switches. For example, the tenant in server h_8 in Figure 4.4 will have three new locations to be considered, that are shown in blue blocks. On the contrary to the constraint programming solution in the Section 4.3, the knapsack problem in fat-tree will optimise for minimising the residual resources as all locations in a new-location valid for a request, will have equal cost.

In the Near-Optimal Subset-sum shown in Algorithm 5, Q is a set of initial requests, each represents a tenant's request to certain module, P is a set of locations available for allocating requests and the set A refers to the set of allocated requests in certain locations, the function $GetNewLocations()$ create new locations schema from the fat-tree while $GetLocations(p^*)$ will return the list of locations in the new_location p^*, $Unallocated(Q, A)$ will return the unallocated requests by searching Q of request that not in allocation A while $GetOptimal(Q', P')$ will solve the knapsack problem for requests Q' and location P'. For requests N and locations M, the complexity of Near-Optimal Subset-sum is $O(MNC)$ where solving 0-1

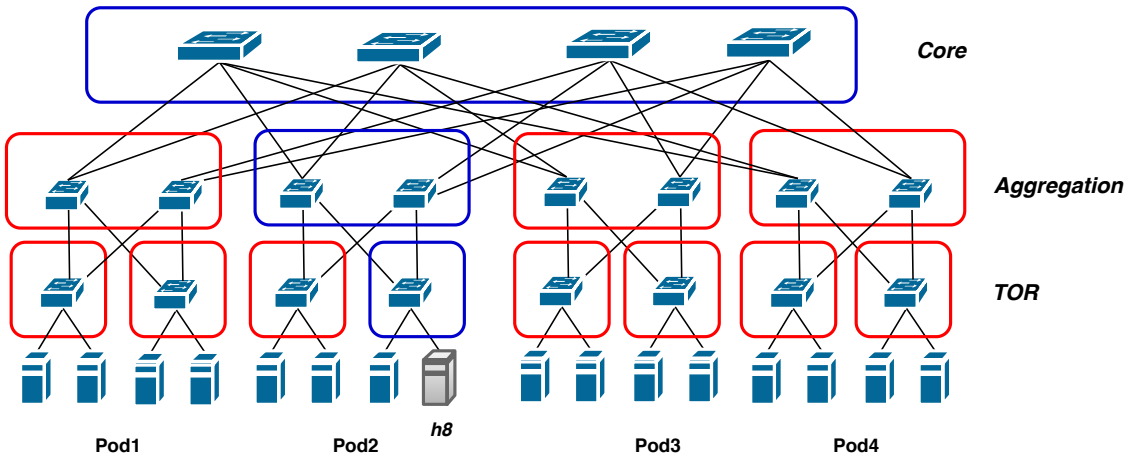


Figure 4.4: New Locations Schema for Fat-Tree Size k=4

Algorithm 5 Near-Optimal Subset-Sum

Input: Set of requests Q, set of locations P
Output: Set of requests allocated to locations A

$A \leftarrow \emptyset$ ▷ initialisation
$Q' \leftarrow Q$
$P^* \leftarrow GetNewLocations()$
for all $p^* \in P^*$ **do**
  $P' = GetLocations(p^*)$
  $A' \leftarrow GetOptimal(Q', P')$
  $A \leftarrow A \cup A'$
  $Q' \leftarrow Unallocated(Q, A)$
return Set of allocated requests A
function GETOPTIMAL(Q, P)
  $cpmodel \leftarrow CreateCpModel()$ ▷ Create model
  $\mathbf{x} \leftarrow AddTwoDimBinaryVariable(cpmodel, Q, P)$ ▷ problem variable
  for all $r \in Q$ **do**
    for all $p \in P$ **do**
      $AddValidConstraint(cpmodel, w(x,p), X(r,p))$ ▷ valid constraint
  for all $r \in Q$ **do**
    $AddLocationConstraint(cpmodel, r)$ ▷ single location per request constraint
  for all $s \in S$ **do**
    $AddSwitchCapacityConstraint(cpmodel, s)$ ▷ switch capacity constraint
  for all $l \in L$ **do**
    $AddLinkCapacityConstraint(cpmodel, l)$ ▷ link capacity constraint
  $comp\_obj \leftarrow CreateCompObj(S, u, Q, P)$
  $residual\_obj \leftarrow CreateResidualObj(Comp\_obj)$
  $AddMinimizeObjective(cpmodel, residual\_obj)$
  $solution = SolveCPModel(cpmodel)$ ▷ Solve model
  $A \leftarrow GetAllocation(solution)$
  return Set of allocated requests A

knapsack problem would take $O(NC)$ where C is the number of capacity slots used in the dynamic programming method; it is, therefore, a constant (a configurable parameter) and it will be solved M times.

4.7 Linear Programming

The One-dimensional implementation is considering the computing resources of the fat-tree data center and eliminates the communication overhead, which converts the problem to a representation that can be solved by Integral linear programming (ILP). To satisfy the traffic constraints, the stateless class modules will independently duplicate as shown in Chapter 3 over links in case of allocations where traffic is distributed over multiple links. However, this model will eliminate the communication overhead that can be endured in the case of a stateful class by allocating the stateful class modules to the ToR switches where they can capture all traffic destined to the requested tenant.

For example, in the k=4 fat-tree shown in Figure 4.4, a stateless class module deployed for a tenant in the server 8 has three available locations (shown in blue blocks) that satisfy the traffic constraints where each allocation will have redundant instances in each switch. On the other hand, a stateful module will be directly deployed at the corresponding ToR switch, which eliminates steering traffic to the instance or inducing communication between distributed instances. Adopting this strategy allows us to eliminate the communication resources in this model.

4.7.1 Locations Rearrangement

To represent the problem as VSBPP in ILP form, every request must be allocated to only one location. Therefore, switches are rearranged in location schema similar to the one adopted by the Near-Optimal solution in Section 4.6 and shown in blocks in Figure 4.4. The locations schema combines switches with duplicated instances in aggregation and core level to one location. For example, all core switches now combined to one location. Consequently, the

number of locations in the structure will be changed, for instance in the k=4 fat-tree shown in Figure 4.4, the number of locations will be reduced to 13 locations. In addition, locations combining more than one switch will have a capacity equal to the sum of the capacity of the combined switches. Also, the cost of the resources for allocating a module to a location with combined switches will be equal to the cost of duplicating the module VNFs across the switches in the new location. For example, a tenant in server 8 in Figure 4.4 will have three valid locations that are shown in blue blocks to allocated stateless modules. The ToR blue block location with a total cost equal the cost of the deployment of one instance. The second location will be the blue block in the aggregation level with the cost of two instances, and the third location in the core level will have a cost of four instances deployment.

4.7.2 One-dimensional VS Two-dimensional

The One-dimensional and two-dimensional models represent two implementations to the placement problem of the security functions with different approaches to the infrastructure resources. The main difference between the two is that the one-dimensional model only considers the computing resources dimension of the problem, while the two-dimensional considers both computing and communication resources of the infrastructure. Therefore, the one-dimensional has no communication overhead where the stateful class functions are allocated to points where there is no split of traffic which are the less consuming resources locations. Subsequently, the stateless functions will have increased probability to be allocated in locations that require full duplicated instances deployment, which increases the total computing resources consumption of the placement. Therefore, the one-dimensional model requires more computing resources than the two-dimensional model to accommodate the same set of requests.

Furthermore, the one-dimensional implementation in a fat-tree architecture would allocate the stateless class functions in the TOR level where there is no traffic split. Therefore, every request will have only one location that can satisfy traffic constraints. Hence, it will reduce the framework fault-tolerance where no replacement location for stateful class functions can

be found in case of TOR location failure. On the other side, the communication overhead in the two-dimensional model can be a problem in case of network congestion. We conclude a trade off between the computing and communication resources of the infrastructure are to be considered in choosing one model over the other.

4.8 Summary

The fat-tree is the most common architecture in virtualised data centers where three layers of switches connect the north-south traffic coming from outside of the network to the tenants' servers. In this chapter, we have presented a carefully designed implementation to the placement framework presented in Chapter 3 for the fat-tree architecture. The implementation determines the location's computing capacity in each level to balance the allocation in the three-level architecture. Furthermore, the implementation of the allocation strategy such as independent duplication, dependent duplication and single-instance are designed to save computing and communication resources in the three-tier architecture. Moreover, the implementation of the constraint programming, heuristic, meta-heuristic and near-optimal solutions in this chapter are modified to fit the characteristics of the fat-tree architecture. For example, heuristic solutions will balance the dependent allocation of the stateful class at a certain level by selecting the worst fit allocation. At the same time, the subset-sum algorithm is designed to optimise the allocation for each location as a knapsack problem to minimise the residual resources where all request's allocations have the same cost in the fat-tree new location schema. Furthermore, a one-dimensional implementation that eliminates the communication overhead of the problem has been designed; however, it causes an increase in the computing cost of the placement compared to the two-dimensional model.

Chapter 5

Evaluation

5.1 Overview

The evaluation shown in this chapter has been selected to show the most important characteristics of the placement framework. In addition to, illustrate the optimality, efficiency and scalability of the developed heuristic, meta-heuristic, near-optimal solutions. First, Section 5.2 shows the objective functions of the resource-aware placement introduced in chapter 3 and the performance metrics used in the evaluation. Section 5.3 depicts the evaluation environment used to evaluate the proposed framework and the placement algorithms. Then, it details the simulation parameters and demonstrates the implemented workloads. In Section 5.4, we compare the optimality of the constraint programming, heuristic, meta-heuristic, near-optimal placement algorithms for the implemented workloads. Furthermore, it compares the proposed framework against the legacy single instance allocation and shows the evaluation of the LP and heuristic solutions for the one-dimensional model. Section 5.5 extends the evaluation of the promising solutions to show the performance metrics for different network sizes. Then, we present the optimality gap analysis and evaluate the execution time and success rate of the algorithms. Then, we evaluate the effect of the class types distribution and number of offered modules. Section 5.6 evaluate the scalability of the proposed algorithms while increasing the request rate. Finally, Section 5.7 concludes the chapter.

5.2 Performance Metrics

To evaluate the resource allocation algorithms, we introduce two metrics that represent our objectives presented in Chapter 3. Furthermore, two metrics represent the speed and success rate performance of the algorithms and the placement ratio metric that represent the utilisation of resources and a final metric to represent the optimality gap.

The first metric is the ***Residual Resources*** **(RS)** of the network, which is the ratio of the spare resources (after placement) to the total amount of resources available and is calculated by adding the residual resources at each location after placement. RS is a normalisation of our first objective presented in Section 3.4.

The second metric is ***Communication Overhead*** **(CO)** of the allocation as the total of the communication overhead resulting from sharing information by the stateful class which represents the sum of communication overhead traffic rate on each link multiplied by the link weight. CO is a normalisation by the total consumed bandwidth to represent our second objective presented in Section 3.4.

The third metric is ***Success Rate*** of the algorithm as the average number of times the algorithm was able to find a feasible solution to the problem.

The fourth metric is ***Execution Time*** of the algorithm, which represents the average of time in seconds spent to reach the final solution to the problem.

The fifth metric is the overall ***Placement Ratio*** **(PR)**, which represents the ratio of allocated resources out of the total amount of resources requested, For example, PR=1 will indicate meeting all requests by finding the allocation that satisfies their constraints, while PR$<$1 indicates a failure ratio where not all requests are satisfied. Furthermore, it indicates the utilisation level of resources, where an efficient algorithm will result in a higher amount of allocated resources.

The sixth metric is the average ***Optimality Gap*** **(G)** of a solution, which represents the difference between the solution and the optimal solution. The optimality gap is calculated using Equation 5.1 Where R is the average value of the algorithm objective, R_{op} is the average

value of the optimal solution objective.

$$G = \frac{R - R_{op}}{R_{op}} \tag{5.1}$$

5.3 Experimental Setup

The evaluation was conducted using software to simulate the static state of fat-tree data centers topology. The software runs on a desktop computer (8GB of RAM, Intel i7). The environment was built using Python 3.6, The CP and LP models were implemented on the IBM ILOG CPLEX [1] using the object-oriented modelling Python API **DOCplex**. The fat-tree network topology builds using The **NetworkX** [2] library. Without loss of generality, we assume traffic is uniformly distributed on all servers, and each server represents one tenant, resources as a one-dimensional vector. The evaluation focuses on the stateless and stateful classes with the independent and dependent duplication strategy respectively. While the single instance and flow-rate control strategy will be forced as hard constraints to the placement process where limited locations satisfy their constraints. All results are computed over an average of 10 runs. The python source code of simulated environment and solutions along with the instructions to replicate the experiments can be found in GitHub[3]. In the next paragraphs, we outline the most important characteristics of the simulated environment.

5.3.1 Simulation Parameters

We assume a *server request capacity* is 100 units as the maximum computing resources requested capacity of a server while links that connect switches and servers have a bandwidth capacity of 100 units with over-subscription upper bound of o=1.5. A tenant requested security services as a set of modules to process the traffic destined for the tenant, represented as tenant request rate r which is the number of functions requested by a tenant. The resources

[1] https://www.ibm.com/products/ilog-cplex-optimization-studio/
[2] https://networkx.github.io/
[3] https://github.com/AbeerFaroukAli/Security_Placement.git

Parameter	Description
k	Fat-Tree Size
o	Over-subscription Upper Bound
r	Tenant Request Rate (Workload)
n	Number of Security Functions
p	Probability of Security Function to be Stateless
μ	Mean of the Normal Distribution
σ	Standard Deviation of the Normal Distribution
$ServerCapacity$	Computing Resources of a Server
$LinkCapacity$	Links Capacity Bandwidth
$TrafficDemand$	Traffic Demand Rate of a Tenant (Workload)
$ModuleSize$	Module Required Resources (Workload)
$CommunicationOverhead$	Sharing information Rate
$TimeLimit$	Time Limit Parameter of the Cplex optimiser
RS	Allocation Residual Resources (Objective)
CO	Allocation Communication Overhead (Objective)
PR	Allocation Placement Ratio
G	Optimality Gap of Algorithm
$SuccessRate$	Success Rate of Algorithm to Find the Solution
$ExecutionTime$	Average of Execution Time for an Algorithm

Table 5.1: Simulation Parameters

requested for each module are drawn for a normal distribution with a maximum of 25% of server maximum request capacity. The traffic demand rate of each tenant is drawn for a normal distribution with a maximum of 100% of link capacity. We simulated n different sizes of security functions. A security function will have a probability p to be stateless and $(1-p)$ to be stateful. For the stateful class module, the communication overhead of sharing information is drawn from a normal distribution with the maximum as the traffic rate between the sharing nodes. The initial capacity of the switches in each level is determined based on the size of the network as detailed in Section 4.2. Simulation parameters, workloads and objectives symbols are listed in Table 5.1.

5.3.2 Workload

We simulated the increase of *workload* over the network in two dimensions: the modules required resources (modules' sizes) and the traffic demand of tenants. The former is represented as the mean of the normal distribution which resources requirements of the modules are drawn from. The latter is represented as the mean of the normal distribution which traffic demand rate of tenants is drawn from. The mean values of the modules required resources shown in the evaluation is a percentage of the maximum value of resources for a module, and the mean values of the traffic demand is a percentage of the maximum value of traffic demand for a tenant which is 100% of the link bandwidth as mentioned above.

5.3.3 The System Capacity Constraint

The system capacity constraint represents the search space of the problem which extend from when the workload is very light (for BFD case: most of the requests will be kept in TOR level) to heavy workload (for BFD case: function allocation will be distributed on the three infrastructure levels) in a condition that *all requests have been allocated*. This constraint will be enforced by keeping 10% to 20% slack percentage in each experiment [146] where a slack percentage is the percentage of total required resources over the total available resources. For the following experiments, the simulation parameters are tuned for each experiment to keep the system capacity constraints satisfied unless otherwise is stated.

5.4 Results Analysis

In this section, we present the evaluation of the placement solutions implemented in Chapter 4. We tested the solutions and compared their performance of residual resources and communication overhead objectives of the security placement problem against different workloads. For a complete evaluation for each solution method, we implemented different variants of each method. For clarity, we divided the tested methods over the following sections.

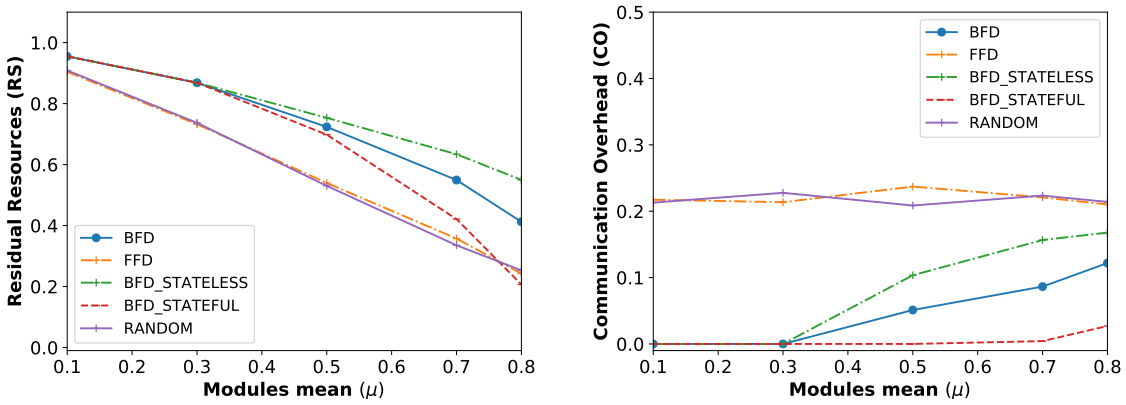
In each, we tested the variants for each method to find the most efficient then compare it with the next section. For each solution method, two experiments have been conducted to show the performance of the tested methods against the workloads presented in the previous section. The **first experiment** shows the *effect of the modules sizes workload* of the tested algorithms and the **second experiment** shows the *effect of traffic demand of tenant workload*. While the simulation parameters are tuned for each experiment to keep the system capacity constraints satisfied.

The first experiment shows the effect of modules' sizes on the tested algorithms on the performance metrics RS and CO in case of traffic demand rate parameters are μ=80% and σ=10% of the maximum value of flow rate, tenant request rate equal r=4, communication overhead parameters are μ=20% and σ=10% of the max value of flow rate between shared nodes, base_part to traffic_part percentage is 50%, number of available modules n=20, and the probability of the stateless class $p = 50\%$. While the second experiment shows the effect of traffic demand workloads on the performance metrics RS and CO in case of modules size (required_resources) parameters are μ=70% and σ=10% of the maximum value of module size and the same parameters for the first experiment.

5.4.1 Heuristic Solutions

We compare the heuristic greedy algorithms BFD, FFD and RANDOM. We also evaluate the three BFD algorithms BFD, BFD_STATELESS and BFD_STATEFUL detailed on Section 4.4. The BFD algorithm with no bias, it sorts requests in non-increasing order based on computing resources required for each while BFD_STATELESS is biased to stateless class functions and the BFD_STATEFUL is to stateful class requests. We present results of the RS and CO for the following experiment:

The results of RS and CO metrics for the first experiment of BFD, BFD_STATELESS, BFD_-STATEFUL, FFD, and RANDOM algorithms for a k=6 fat-tree are shown in Figure 5.1. Figure 5.1a shows the *objective function* Residual Resources (RS) starts decreasing linearly with the workload, where the increase of requested resources will result in a reduction in



(a) Residual Resources

(b) Communication Overhead

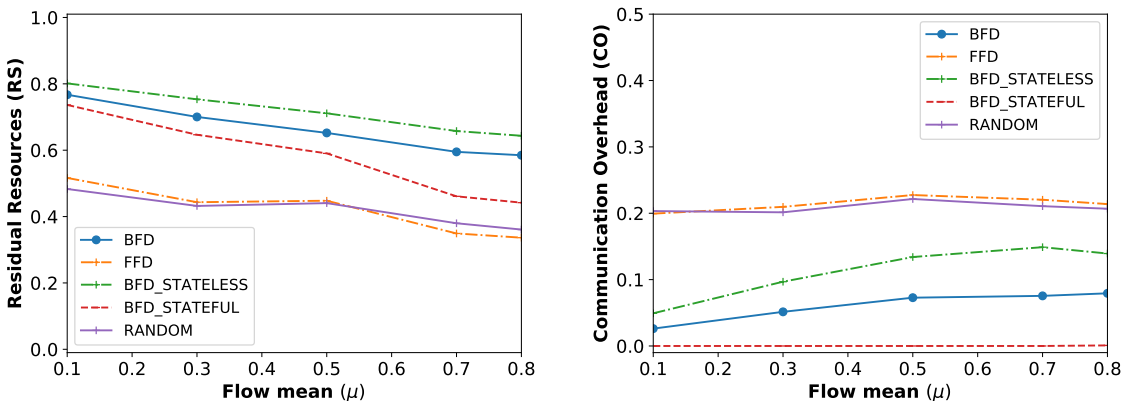
Figure 5.1: RS and CO of Heuristic Algorithms for Modules Sizes Workload, when k=6

spare resources. While all algorithms suffer from such reduction, the Fest-Fit Decreasing BFD algorithms show less reduction in RS than (FFD and RANDOM). This can be attributed to best-fit algorithms utilise resources by selecting locations which cost the least increase in resource consumption, allowing more resources to the allocation process, and leading to an increase in RS that can reach 100% of other algorithms. Furthermore, The figure shows at low workload (less than 0.3) the three BFD algorithms have the same residual resources where all requests have been allocated to the TOR level. However, as workload increases, the BFD_STATELESS algorithm has more residual resources among other BFD algorithms as it gives priority to stateless class function and as they the most consuming function to the computing resources with full duplication, it utilises their allocation and results in reducing overall resource consumption. While BFD_STATEFUL show the least residual resources among all BFD versions as it utilises allocation for the stateful class which less computing resources consumption compared the stateless class.

Furthermore, at higher workload (beyond 0.7) BFD_STATEFUL show slightly less residual resources than FFD and RANDOM algorithms due to increasing percentage of stateless class allocated to higher-layer which results in less residual resources than allocating all classes in random. Moreover, the BFD algorithm shows more residual resources than BFD_STATEFUL but less residual resources than BFD_STATELESS, which the result of it sort requests regardless of their class type and consequently utilise allocation of both of them. Nonethe-

less, it shows more RS by 60% than FFD and RANDOM.

Figure 5.1b shows the Communication Overhead (CO) *objective function* after the allocation is complete and, the results show that at low workload, less than 0.3, best-fit algorithms BFD show no communication overhead while FFD and RANDOM show overhead. This is a result of the FFD and RANDOM allocate requests in random locations which will result in allocations in higher layers of the network which cause a communication overhead while best-fit algorithm allocates requests to a less resource-consuming location first which are TOR switches which have no communication overhead. In higher workload, beyond 0.3, CO is linear with the workload which indicates that more requests are allocated to higher layers which impose a communication overhead of shared information between main and monitoring instances of the stateful class. However, FFD and RANDOM show steady CO as all layers randomly selected with the same percentage of requests which impose the same CO percentage. Moreover, BFD_STATEFUL algorithms show less communication overhead over BFD and BFD_STATELESS where it utilises stateful class functions which the cause the of the communication overhead while it reaches up to 80% less CO over FFD and RANDOM algorithms. In addition to that, the BFD algorithm shows more communication overhead than the BFD_STATELESS but still shows up to 70% less CO than the FFD and RANDOM algorithms.



(a) Residual Resources

(b) Communication Overhead

Figure 5.2: RS and CO of Heuristic Algorithms for Traffic Demand Workload, when k=6

The results of RS and CO metrics for the second experiment for a k=6 fat-tree are shown

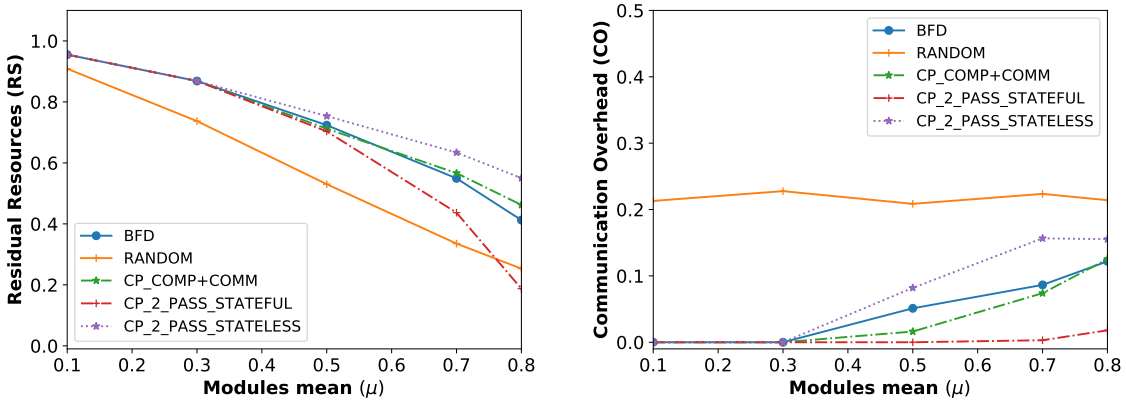
in Figure 5.2. Figure 5.2a shows residual resources decreases with the workload, where the increase of traffic rate results in increasing of required resources as the traffic part of functions' required resources is depending on traffic rate and results in a reduction in spare resources. While all algorithms suffer from such reduction, BFD algorithms still show less reduction than (FFD and RANDOM algorithms) that reach up to more than 60% in spare resources. Moreover, BFD still shows more residual resources than BFD_STATEFUL and less residual resources than BFD_STATELESS the same as in Figure 5.1a with no less than 50% more RS than (FFD and RANDOM algorithms).

In Figure 5.2b, the results show that CO has a linear increase in communication overhead for the BFD algorithms at low workload (less 0.5) due to increasing the flow rate will increase the requested computing resources, and more requests will be allocated to higher layers which cause more communication overhead to the allocation. While at high workload, beyond 0.5, there is a slight increase in CO as a result of, CO is normalised to the total consumed bandwidth which also increases with the workload. On the other hand, FFD and RANDOM algorithms show less increase in CO with the workload increase than other BFD algorithms show and this can be attributed to as all locations start to fill up, FFD and RANDOM will allocate more requests to lower layers of the network which impose no communication overhead to the system. Furthermore, BFD_STATEFUL demonstrates less CO than other BFD algorithms, while BFD_STATELESS shows more CO than BFD. Moreover, BFD has less CO up to 70% of RANDOM and FFD.

We conclude that the BFD algorithm demonstrates higher utilisation of the computing and the communication resources over other greedy algorithms such as FFD or RANDOM algorithm. However, as the sorting phase of BFD change algorithm bias to one of the resources over the other, we select BFD algorithms as our proposed greedy algorithm to solve the placement problem where is balance the allocation of the stateless and the stateful class rather than priorities one of them such as in BFD_STATELESS or BFD_STATEFUL algorithms. Therefore, the BFD algorithm is going to be adopted as the heuristic solution for the remaining of this chapter.

5.4.2 Constraint Programming

The modelled constraint programming solution is used as a baseline for comparison with other solutions. We modelled three variants of constraint programming solution, the CP_COMP+COMM model and the two versions of TWO_PASS model, the CP_2_PASS_STATELESS and the CP_2_PASS_STATEFUL introduces in Section 4.3. While the constraint programming optimiser takes days to solve a single instance of the problem, it stops getting better solutions after a certain time elapses. Thus, we set the $TimeLimit$ parameter of the optimiser to enough time to get the best solutions. We include the result of heuristic BFD and RANDOM algorithms for comparison. We present the results of the RS and CO for the following experiment:



(a) Residual Resources

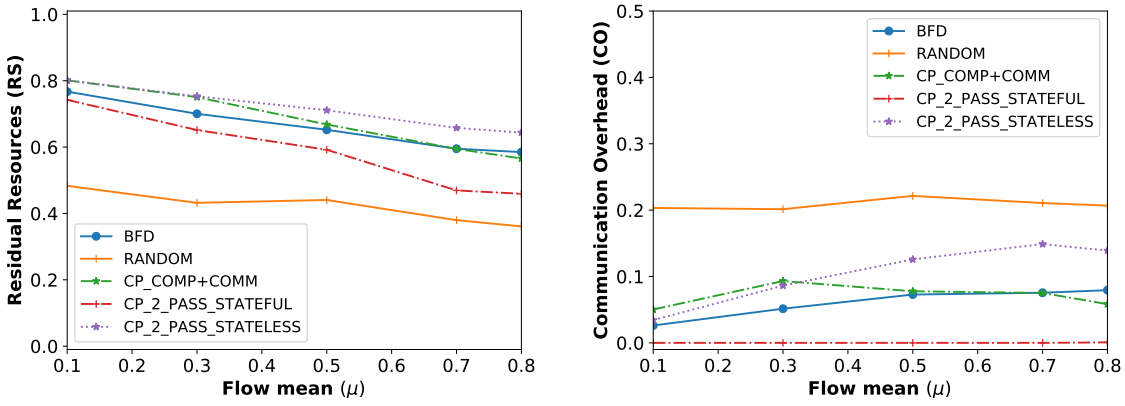
(b) Communication Overhead

Figure 5.3: RS and CO of Constraint Programming Models for Modules Sizes Workload, when k=6

The results of RS and CO metrics for the first experiment of BFD, CP_COMP+COMM, CP_2_PASS_STATELESS, CP_2_PASS_STATEFUL, and RANDOM with $TimeLimit$=10min for a k=6 fat-tree are shown in Figure 5.3. Figure 5.3a RS starts decreasing linearly with the workload for all algorithms, where the increase of requested resources will result in a reduction in spare resources. At low workload (less 0.3) all algorithms are showing the same RS where all requests are allocated to TOR level. However, at high workload (beyond 0.3) CP_2_PASS_STATELESS demonstrate more RS that can reach 60% more than (FFD and RANDOM) algorithms which can contribute to it prioritises the STATELESS class functions

allocation which are the most computing resources consumption. On the contrary, CP_2_PASS_STATEFUL shows the least RS after RANDOM as it prioritises utilising STATEFUL class functions allocation. Furthermore, the results show CP_COMP+COMM the same RS as BFD. However, at a high workload, beyond 0.7, CP_COMP+COMM shows a slightly more RS than BFD by 5%.

Figure 5.3b shows CO after the allocation is complete and, it shows an increase in CO as Figure 5.1b with no communication overhead for algorithms in low workload (less 0.3) except the RANDOM algorithm. Furthermore, it shows that CP_2_PASS_STATEFUL has the least communication overhead as it prioritises the STATEFUL class functions allocation which introduce the communication overhead. On the contrary, CP_2_PASS_STATELESS shows the highest CO after RANDOM algorithm. The results show CP_COMP+COMM has less communication overhead than CP_2_PASS_STATELESS but higher than CP_2_PASS_STATEFUL the same as BFD. However, CP_COMP+COMM shows less CO than BFD that reach up 10% while it shows the same as BFD at high workload.



(a) Residual Resources

(b) Communication Overhead

Figure 5.4: RS and CO of Constraint Programming Models for Traffic Demand Workload, when k=6

The results of RS and CO metrics for the second experiment of BFD, CP_COMP+COMM, CP_2_PASS_STATELESS, CP_2_PASS_STATEFUL, and RANDOM for a k=6 fat-tree are shown in Figure 5.4. Figure 5.4a shows residual resources decreases with the workload as in Figure 5.2a for all algorithms. Furthermore, CP_2_PASS_STATELESS still shows more RS

than other algorithms while CP_2_PASS_STATEFUL shows the less RS than BFD or other CP solutions but more that RANDOM algorithm. In Figure 5.4b, the results show that CO increases as in Figure 5.2b. Furthermore, it shows CP_2_PASS_STATELESS has more CO than other algorithms but less than the RANDOM algorithm while CP_2_PASS_STATEFUL shows less CO than the others. Moreover, the results show that in workload less than 0.7, CP_COMP+COMM has more RS than BFD as shown in Figure 5.4a while it has more CO as shown in Figure 5.4 which can be attributed to it balance the two performance metrics based on their normalised value and in this case RS has outweighed CO and results in RS optimised over CO. However, beyond 0.7 workload, CO is optimised over RS where it shows lower CO than BFD but less RS.

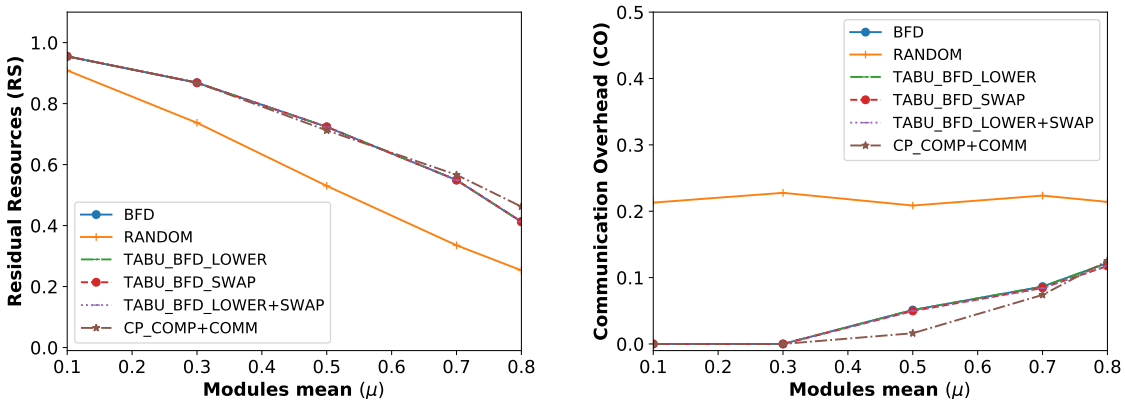
We conclude that BFD and CP_COMP+COMM are demonstrating a balance in optimising both RS and CO more than CP_2_PASS_STATELESS and CP_2_PASS_STATEFUL. Therefore, CP_COMP+COMM will be used as a baseline solution for the remainder of the chapter.

5.4.3 Meta-Heuristic

We implement three variants of the TABU algorithm with different moves types as illustrated in Section 4.5. The algorithms are TABU_LOWER , TABU_SWAP, and TABU_LOWER+SWAP each has been tested with an initial solution of a RANDOM or a BFD algorithm. We include the heuristic BFD and RANDOM and the constraint programming CP_COMP+COMM for comparison.

The results of RS and CO metrics of BFD , TABU_LOWER , TABU_SWAP , TABU_LOWER+SWAP, and CP_COMP+COMM in case of the tabu list length is equal the number of requests to allow a move for every request. for a k=6 fat-tree are shown in Figure 5.5 and Figure 5.6.

Figure 5.5a shows RS and CO of Tabu algorithms TABU_BFD_LOWER, TABU_BFD_SWAP and TABU_BFD_LOWER+SWAP with BFD algorithm as an initial solution. It shows the RS objective function starts decreasing linearly with the workload as expected, similar to Figure 5.1a. Furthermore, it shows that Tabu algorithms have the same RS as the heuris-



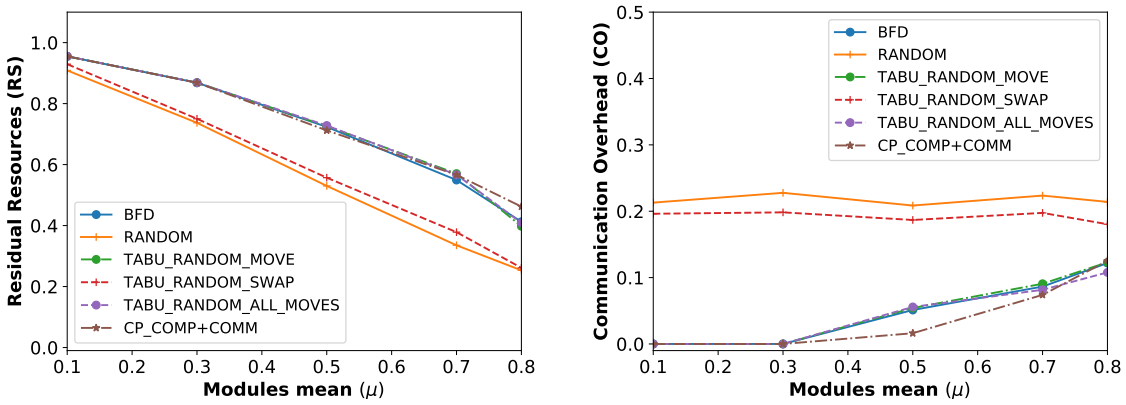
(a) Residual Resources

(b) communication Overhead

Figure 5.5: RS and CO of BFD Meta-heuristic for Modules Sizes Workload, when k=6

tic BFD. Figure 5.5b shows the CO objective function after the allocation is complete and, it shows similar results as Figure 5.1b where algorithms have a linear increase of CO with the workload. It also shows all BFD based Tabu algorithms have the same CO as the BFD heuristic algorithm.

Figure 5.6a shows the Tabu algorithms TABU_RANDOM_LOWER, TABU_RANDOM_-SWAP and TABU_RANDOM_LOWER+SWAP with an initial solution of the RANDOM algorithm. The results show that RS objective function starts decreasing linearly with the workload as expected. Furthermore, TABU_RANDOM_SWAP algorithm has slightly more RS than RANDOM but less than other algorithms which can be contributed to that only swap moves are allowed and start with requests in random locations will result in requests end up in higher layers of the network which cause less residual resources. However, TABU_RANDOM_LOWER and TABU_RANDOM_LOWER+SWAP will improve its allocation by moving requests to lower locations and end up with the same residual resources as BFD. Furthermore, it shows that TABU_RANDOM_LOWER and TABU_RANDOM_-LOWER+SWAP do not show any significant improvement compared to BFD in low workload. However, in high workload (beyond 0.6), TABU_RANDOM_LOWER and TABU_-RANDOM_LOWER+SWAP show very slightly higher residual resources. This can be attributed to, starting by random solution, swap and lower moves can save some RS where it can fill some fragmentation that is resulting from greedy algorithms such as BFD.



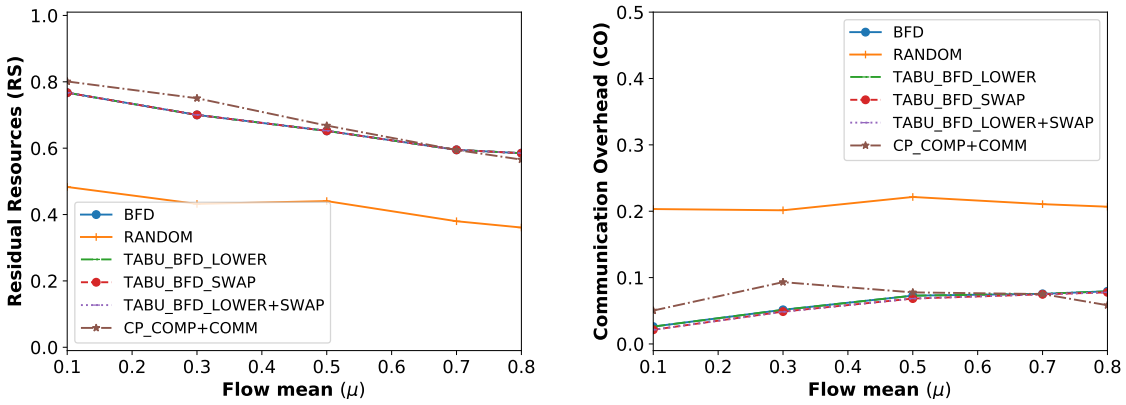
(a) Residual Resources

(b) communication Overhead

Figure 5.6: RS and CO of Random Meta-heuristic for Modules Sizes Workload, when k=6

Figure 5.6b shows the CO objective function after the allocation is complete. It shows that TABU_RANDOM_SWAP algorithm has less CO than RANDOM algorithm but more than the other algorithms which can be attributed to limited swap move restriction. Moreover, it shows that TABU_RANDOM_LOWER+SWAP and TABU_RANDOM_LOWER have the same CO as the BFD algorithm. However, at high workload (beyond 0.7) where TABU_RANDOM_LOWER+SWAP shows a very slight less CO than other algorithms. On the other hand, Tabu algorithms with swap moves (TABU_RANDOM_SWAP and TABU_RANDOM_LOWER+SWAP) take a significant time to finish and that time increases exponentially with network sizes as the number of requests available of the swap is increasing. Therefore we only consider TABU algorithms with BFD as an initial solution for the remainder of experiments.

The results of RS and CO metrics of the second experiment of the Tabu algorithms TABU_BFD_LOWER, TABU_BFD_SWAP and TABU_BFD_LOWER+SWAP with BFD algorithm as an initial solution for a k=6 fat-tree are shown in Figure 5.7. Figure 5.7a shows that residual resources decrease with the workload and that all Tabu algorithms have the same RS as BFD. While Figure 5.7b shows CO after the allocation is complete and the results show that the Tabu BFD algorithms show the same CO as BFD. We conclude that Tabu algorithms that start with BFD can not achieve better utilisation to resources while Tabu that starts with a random solution can achieve slightly higher utilisation. Furthermore, the BFD



(a) Residual Resources

(b) Communication Overhead

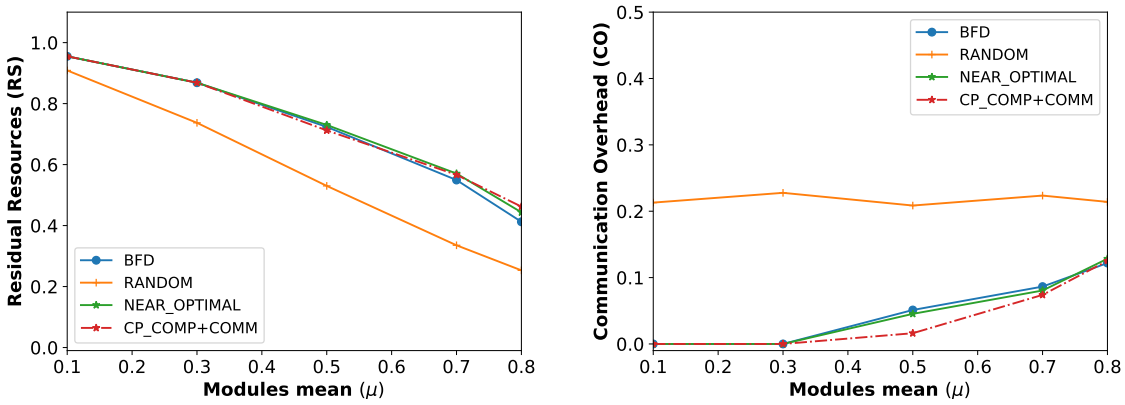
Figure 5.7: RS and CO of BFD Meta-heuristic for Traffic Demand Workload, when k=6

algorithms that start with random algorithms are time consuming while swap based tabu algorithms have exponential complexity with the number of requests.

5.4.4 Subset-Sum Near-Optimal Solution

We evaluated the subset-sum NEAR_OPTIMAL solution presented in Section 4.6. The sub-optimal solutions of the locations are taking a long time to finish without getting any better solutions the same as the constraint programming solution. Therefore, the $TimeLimit$ parameter is set to terminate the search. We include the results of the heuristic BFD and RANDOM and constraint programming CP_COMP+COMM for comparison. We present results of the RS and CO for the following experiment:

The results of RS and CO metrics of the first experiment of BFD, CP_COMP+COMM, NEAR_OPTIMAL, and RANDOM algorithms in case of $TimeLimit$=$5min$ for a k=6 fat-tree are shown in Figure 5.8. Figure 5.8a show RS decreasing linearly with the workload the same as in Figure 5.1a, and that CP_COMP+COMM algorithm shows a higher RS than other algorithms in high workload as Figure 5.3a. Furthermore, the results show that the BFD algorithm has residual resources that almost overlap with NEAR_OPTIMAL solution. However, In high workload (beyond 0.6) NEAR_OPTIMAL was able to spare more residual resources than BFD up to 5% which the same RS as the optimal CP_COMP+COMM which can be at-

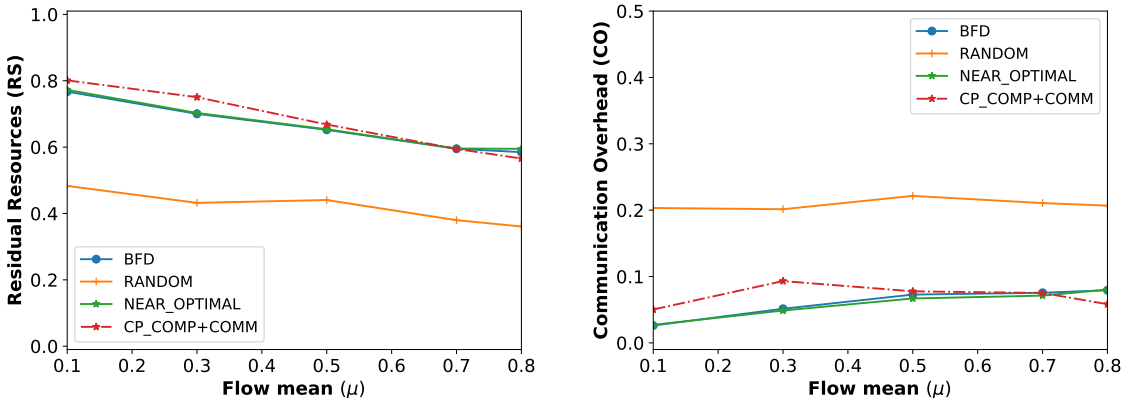


(a) Residual Resources (b) communication Overhead

Figure 5.8: RS and CO of Near Optimal for Modules Sizes Workload, when k=6

tributed to NEAR_OPTIMAL is optimising to minimise computing resources will minimise the resources wasted from fragmentation seen in BFD and increase residual resources.

Figure 5.8b show the CO objective function after the allocation is complete and, the results show that CO of CP_COMP+COMM has less CO the same as shown in Figure 5.3b. Moreover, the results show that communication overhead of the BFD algorithm is almost overlapping with NEAR_OPTIMAL solution.



(a) Residual Resources (b) Communication Overhead

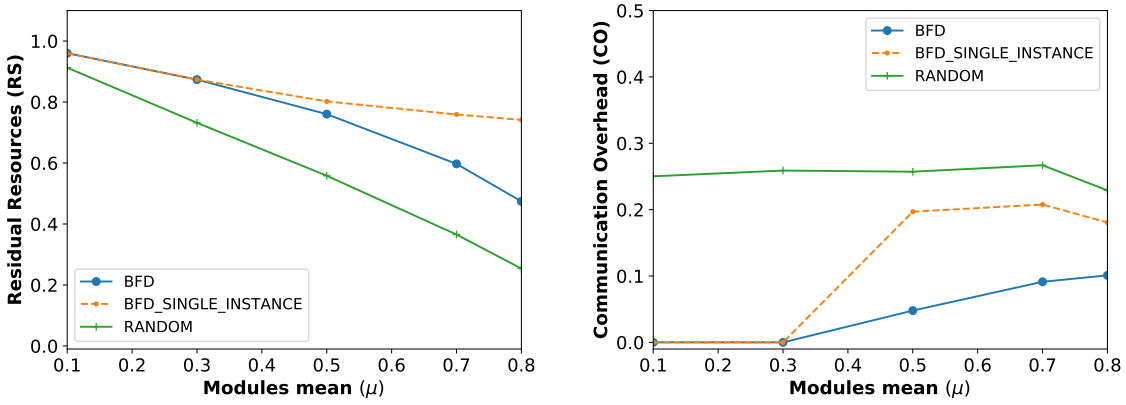
Figure 5.9: RS and CO of Near Optimal for Traffic Demand Workload, when k=6

The results of RS and CO metrics for the second experiment for a k=6 fat-tree are shown in Figure 5.9. Figure 5.9a shows a decrease in residual resources with the workload as expected similar to Figure 5.2a as a result of increasing computing resources with traffic

rate. Furthermore, similar to Figure 5.8a, in high workload (beyond 0.8) NEAR_OPTIMAL was able to spare slightly more residual resources than BFD. Figure 5.9b shows the CO objective function after the allocation is complete. It shows that NEAR_OPTIMAL has the same CO as BFD. We conclude that NEAR_OPTIMAL can reach a slightly better usage of the computing and the communication resources than the BFD algorithm.

5.4.5 Single Instance Allocation

We compare the greedy algorithm BFD to the middleboxes single instance legacy allocation. A single instance is where a request is allocated to a single instance of the requested function, and all traffic must be steered to that instance. For comparison reasons, we adopt the BFD version of the algorithm, the BFD_SINGLE_INSTANCE. Because BFD_SINGLE_INSTANCE is a high bandwidth consumption algorithm, we changed the experiment parameters of the two experiments. Otherwise, it would not be able to find a complete solution to the problem being solved. We present results of the RS and CO for the following experiments:



(a) Residual Resources

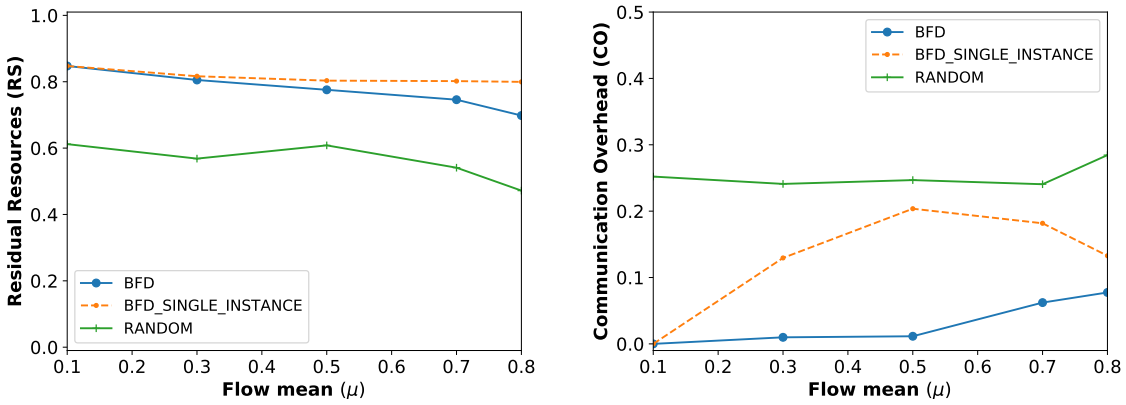
(b) communication Overhead

Figure 5.10: RS and CO of Single Instance for Modules Sizes Workload, when k=6

The results of RS and CO metrics for the first experiment of the BFD, BFD_SINGLE_INSTANCE, and RANDOM algorithms in case of traffic demand rate parameters are μ=60% and σ=10% of the maximum value of flow rate, and over-subscription upper bound are raised to o=2 for a k=6 fat-tree are shown in Figure 5.10. In Figure 5.10a, RS starts decreasing lin-

early with the workload as in Figure 5.1a. The results show that BFD_SINGLE_INSTANCE algorithm has more spare resources than other algorithms which are a result of only one instance being deployed for each request, and no duplication is endured. It shows up to 50% more residual resources than BFD. Figure 5.10b show the CO objective function after the allocation is complete and, the results show, opposite to RS, BFD_SINGLE_INSTANCE shows a linear increase in CO as the traffic steered to the centralised instance will cause a significant communication overhead compared to the distributed allocation strategy adopted by other algorithms where BFD can reach to 50% less CO than BFD_SINGLE_INSTANCE.

The results of RS and CO metrics for the second experiment in case of μ=50% and σ=10% of the maximum value of module size and o=2 for a k=6 fat-tree are shown in Figure 5.11. Figure 5.11a shows a slight decrease in residual resources with the workload as similar as a result of increasing computing resources with the traffic rate. Furthermore, similar to Figure 5.10a, Figure 5.2a shows that BFD_SINGLE_INSTANCE algorithm has more spare resources as a result of single instance deployment policy. While Figure 5.11b results show a high increase of CO for BFD_SINGLE_INSTANCE that reach up to 90% more CO than BFD.

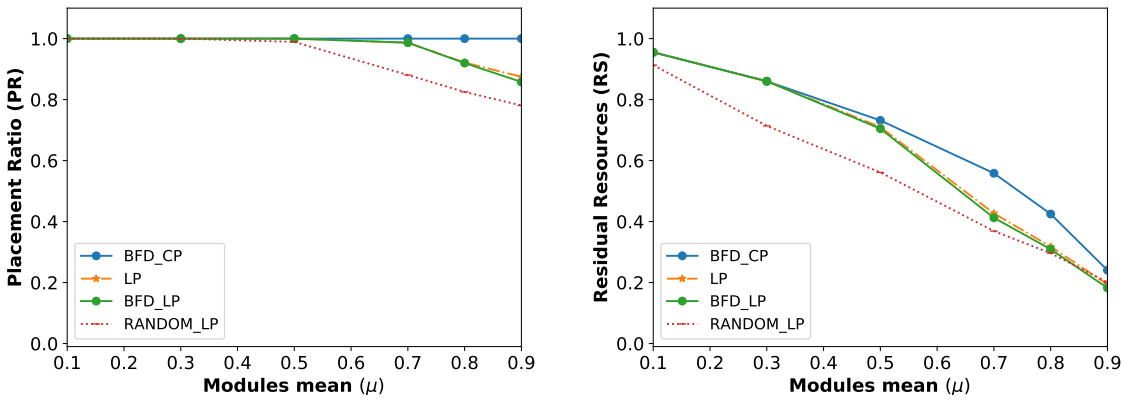


(a) Residual Resources

(b) Communication Overhead

Figure 5.11: RS and CO of Single Instance for Traffic Demand Workload, when k=6

We conclude that BFD_SINGLE_INSTANCE algorithm has more spare computing resources than BFD algorithms, while it suffers a very high communication overhead due to the steering traffic policy.



(a) Placement Ratio (b) Residual Resources

Figure 5.12: PR and RS of BFD_CP and BFD_LP Algorithms ,when k=6

5.4.6 Linear Programming

The evaluation shown in this section illustrates the characteristics of the one-dimensional implementation of the placement framework. It demonstrates the optimality of the heuristic solution against the LP solution of the implementations and compares the one-dimensional model to the two-dimensional model. We compare the BFD algorithm of the one-dimensional model BFD_LP to the LP solution and the BFD of the two-dimensional model BFD_CP. Furthermore, we implemented RANDOM versions of the one-dimensional model RANDOM_-LP. The LP solution has been modelled in MIP CPLEX optimiser.

For the comparison with the two-dimensional implementation, we extend the experiment to include the case where not all requests have been allocated. Therefore, we add an extra virtual location to the actual locations in the one-dimensional implementation with unlimited capacity. Besides, any request that is allocated to that location will be considered as an unallocated request in the final solution. However, the cost of allocating a request to that location is set to more than the cost for Core layer allocation, and proportional to the required computing resources of that request.

The experiment shows the *effect of the modules sizes workload* of BFD_CP, BFD_LP, LP and RANDOM_LP where the traffic demand rate parameters are μ=80% and σ=10% of the maximum value of flow rate, tenant request rate equal r=4, communication overhead

parameters are μ=20% and σ=10% of the max value of flow rate between shared nodes, base_part to traffic_part percentage is 50%, the number of available modules n=20, and the probability of the stateless class is p=50%. The results of PR and RS metrics for a k=6 fat-tree are shown in Figure 5.12. It is worth mentioning that the following experiment is not satisfying the system capacity constraint.

Figure 5.12a shows the placement ratio near 1 for low workloads. While beyond 0.5 workload, PR starts decreasing linearly with the workload for LP-based algorithms, which occurs as a result of the requested resources starting to exceed resource capacity available in switches as a result of the stateful class constraint will reduce available resources for the stateless class which are the most consuming type for resources. Moreover, the results show that the BFD_CP has PR=1 While BFD_LP suffers a decrease in PR, it shows less reduction than RANDOM and overlap with the LP solution. While Figure 5.12b shows the residual resources after the allocation is complete and, similar to the PR case, it exhibits a reduction as workload increases where the increasing of requested resources will result in a reduction in spare resources. Furthermore, it shows that when $workload$>0.5 BFD_LP has more residual resources than the RANDOM algorithm and almost overlaps with the LP algorithm. Furthermore, the BFD_LP and LP algorithms have less residual resources to spare than BFD_CP algorithm that reach 20% even if the PR is less than 1, and it could not accommodate all requests.

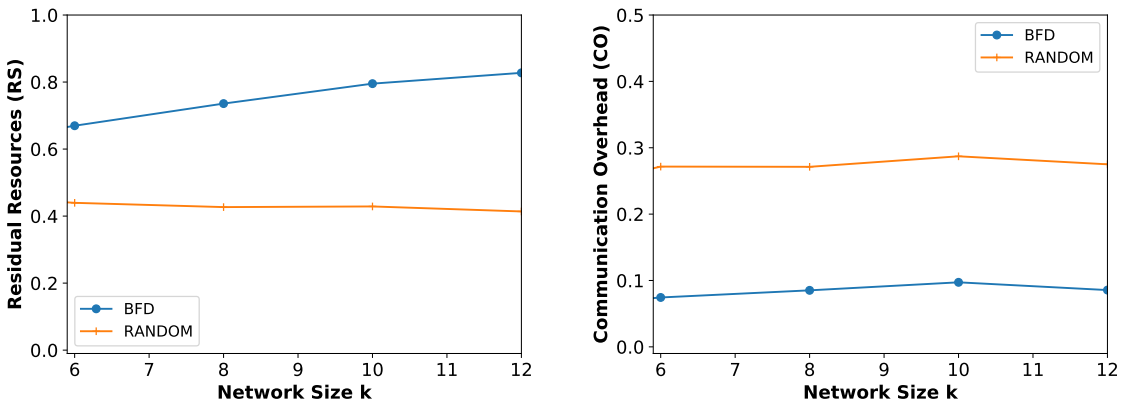
Based on the above results, the BFD_P exhibits better resource utilisation to accommodate more resources compared to the RANDOM algorithm and show near-optimal results compared to the LP solution. However, BFD_LP has less residual resources than BFD_CP, where stateless class consuming more resources in higher layers than stateful class functions. Therefore, the LP implementation of the framework will require more computing resources to match the CP model accommodation level for requests. To accomplish that for a fat-tree, it will require an update to the capacity of the deployment locations presented in Section 4.2.

5.5 Extended Analysis

In this section, we extend the evaluation of the framework and the developed algorithms. First, we evaluate the framework resources capacity presented in Section 4.2 in different network sizes. Second, we present the optimality gap analysis of the methods that show promising performance in the previous section. Then, we evaluate the efficiency of the algorithms for execution time and success rate. Finally, we evaluate the effect of the class types distribution and the number of offered modules on the algorithms.

5.5.1 Network Size

The next experiment shows the effect *Network Sizes* on the residual resources and communication overhead of the BFD and RANDOM algorithms at low workload where modules size (required_resources) parameters are μ=60% and σ=10% of the maximum value of module size and traffic demand rate parameters are μ=60% and σ=10% of the maximum value of flow rate, tenant request rate r=4, communication overhead parameters are μ=20% and σ=10% of the max value of flow rate between shared nodes, base_part to traffic_part percentage is 50%, the probability of the stateless class p=50%, and the number of modules n=20. fat-tree with $k \in \{6, 8, 10, 12\}$ were tested. The results of RS and CO metrics for tested networks are shown in Figure 5.13.



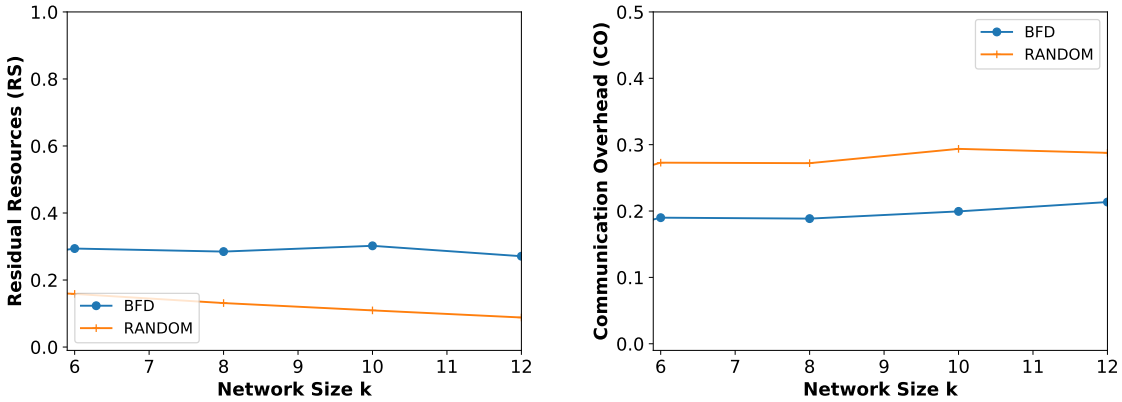
(a) Residual Resources

(b) Communication Overhead

Figure 5.13: RS and CO for Network Size, in Low Workload

Figure 5.13a shows RANDOM has less residual resources than BFD as Figure 5.1. Furthermore, it shows that the RANDOM algorithm has steady residual resources as network size increases which results of requests are allocated randomly to the three layers with equal percentage. However, the results show RS for BFD increasing linearly with the network size which can be attributed to that in low workload, BFD will keep requests in lower layers to reduce duplication while computing resources available in higher layers location is increasing exponentially with network size as illustrated in Section 4.2 which results in more residual resources percentage in higher network sizes. While Figure 5.13b shows that BFD and RANDOM have a steady CO with network size.

The next experiment shows the effect *Network Sizes* on the residual resources and communication overhead of the BFD and RANDOM algorithms at high workload where modules size (required_resources) parameters are μ=90% and σ=10% of the maximum value of module size and traffic demand rate parameters are μ=70% and σ=10% of the maximum value of flow rate and the same parameters for the previous experiment. The results of RS and CO metrics for tested networks are shown in Figure 5.14. Figure 5.14a shows the scalability of the BFD algorithm in a high workload where RS percentages are stable through all network sizes While Figure 5.14b shows the same for communication overhead which prove the the switches capacity selected in Section 4.2.3 is able to scale with network size to accomplish the same level of request satisfaction.



(a) Residual Resources

(b) Communication Overhead

Figure 5.14: RS and CO for Network Size, in High Workload

μ	Heuristic			Meta-heuristic (TABU)			NEAR_OPTIMAL
	BFD	FFD	RANDOM	LOWER	SWAP	LOWER+SWAP	
0.1	0.00	5.96	5.77	0.00	0.00	0.00	0.00
0.3	0.00	2.66	2.73	0.00	0.00	0.00	0.00
0.5	0.08	1.29	1.23	0.08	0.07	0.07	0.04
0.7	0.06	0.70	0.75	0.06	0.05	0.05	0.00
0.9	0.07	0.46	0.45	0.07	0.06	0.06	0.03

Table 5.2: Optimality Gap, when k=6

5.5.2 Optimality Gap Analysis

The optimality gap analysis measures the ratio of how close the provided solutions are to the optimum. The experiment in this section shows the optimality gap by comparing the objective function result of a solution to the constraint programming solution. The optimality gap of an algorithm is calculated using Equation 5.1 Where R_{op} is the average value of the objective of the CP_COMP+COMM solution.

The results of optimality gap G metric of BFD, FFD, RANDOM, TABU_LOWER, TABU_-SWAP, TABU_LOWER+SWAP and NEAR_OPTIMAL algorithms in case of same parameters as the first Experiment of previous section for a k=6 fat-tree are shown in Table 5.2. It is worth mentioning that the Tabu algorithms have a BFD initial solution. The results show that the average optimality gap in case of μ of the size of the modules is in $[0.1, 0.3, 0.5, 0.7, 0.9]$. The results show that the heuristic BFD and meta-heuristic solutions can provide solutions close to the optimal constraint programming solution where the maximum gap was 0.08% while RANDOM and FFD algorithms show the highest gap that can reaches up to 5.96%. Furthermore, results show NEAR_OPTIMAL algorithm has the least gap among all algorithms with a maximum of 0.04%, which underlines a higher accuracy than other algorithms.

5.5.3 Execution Time

While the optimality gap results show that BFD, meta-heuristic and NEAR-OPTIMAL algorithms show close results to the constraint programming solution; however, their execution

time may vary. Therefore, the next experiment shows the *Execution Time* of BFD, RANDOM, TABU_LOWER, TABU_SWAP, TABU_LOWER+SWAP, NEAR_OPTIMAL and BFD_SINGLE_INSTANCE algorithms. The experiment tested in case of the same parameters as the previous experiment. The results of the tested algorithms for k=6 fat-tree are shown in Figure 5.15. It worth-mentioning that the results of BFD_SINGLE_INSTANCE shown are with allocating percentages that are less than 1, which means not all requests have been allocated due to the high bandwidth requirement of the algorithm.

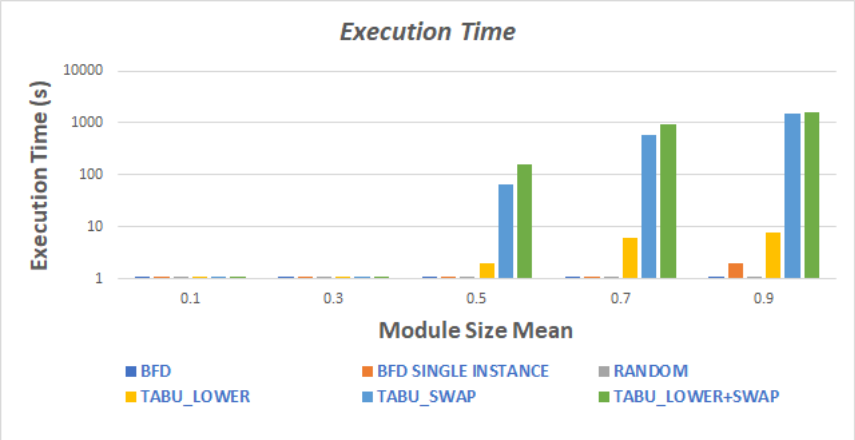


Figure 5.15: Execution Time for Modules Sizes Workload, when k=6

Figure 5.15 shows that at low workload, less 0.3, execution time is less than one second for all algorithms while BFD and RANDOM algorithm keep the 1 second execution time for all workloads. The results also show that TABU algorithms have increasing execution time where an increasing number of requests are allocated in different layers and will be available for swapping. However, TABU_LOWER has less execution time than other TABU algorithms where it reaches more than 10 seconds at 0.9 workload while TABU_SWAP and TABU_LOWER+SWAP have shown a high execution time in a high workload where they exceed 1300 second at workload 0.9. Besides, the results show that BFD_SINGLE_INSTANCE show execution time of 3 seconds at workload 0.9. It is worth mentioning that we omit the results of NEAR_OPTIMAL and CP algorithms as their execution time is determined by the time limit set by the experiment. We conclude that the BFD algorithm outperforms TABU and NEAR_OPTIMAL algorithms in execution time while they slightly outperform BFD in optimality.

5.5.4 Success Rate

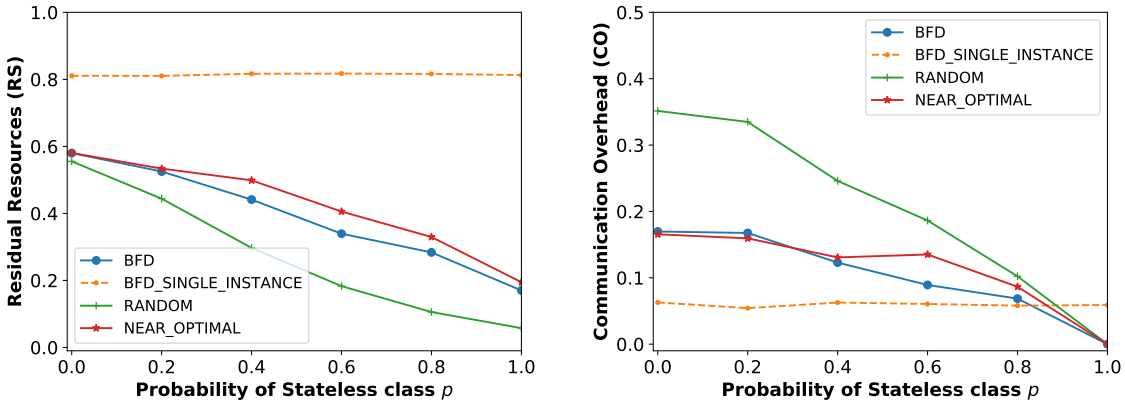
In some cases, an algorithm could not find a solution to the problem. Thus, the next experiment shows *Success Rate* of BFD, BFD_SINGLE_INSTANCE, NEAR_OPTIMAL and RANDOM algorithms. It is worth mentioning that this experiment is tested with search space that violates the system capacity constraint where μ of the modules size is $[0.1, 0.3, 0.5, 0.7, 0.9]$ and traffic demand rate parameters are μ=70% and σ=10% of maximum value of flow rate, tenant request rate equal r=4, communication overhead parameters are μ=20% and σ=10% of maximum value of flow rate between shared nodes, base_part to traffic_part percentage is 90%, number of available modules n=20, and the probability of the stateless class p=50%. The results for k=6 fat-tree over 50 runs are shown in Figure 5.16. It is worth mentioning that we omit the Tabu algorithms as they already have an initial solution to start with.

Figure 5.16: Success Rate for Modules Sizes Workload, when k=6

Figure 5.16 shows that at low workload, all algorithms have a success rate of 1 where all algorithms have succeeded in finding a feasible solution. However, the success rate starts decreasing as the workload increases. The results show the BFD_SINGLE_INSTANCE and RANDOM algorithms start desecrating at workload 0.4 and 0.7 respectively, while BFD starts at 0.8 and NEAR_OPTIMAL at 1. Furthermore, BFD_SINGLE_INSTANCE shows zero success rate at workload 0.5 while RANDOM algorithms could not find any solution at workload 0.8. Moreover, BFD was able to find a solution in 40% of the scenario at workload 0.9 while NEAR_OPTIMAL could find a solution to all of them. No algorithms could find any solutions to any of the instances at workload 1.

5.5.5 Class Types Distribution

The next experiment shows the *effect of the probability of the two classes of security functions* of the BFD, BFD_SINGLE_INSTANCE, NEAR_OPTIMAL and RANDOM algorithms on the residual resources and communication overhead in case of modules size (required_resources) parameters are μ=80% and σ=10% of the maximum value of module size and traffic demand rate parameters are μ=80% and σ=10% of the maximum value of flow rate, tenant request rate r=4, communication overhead parameters are μ=20% and σ=10% of the maximum value of flow rate between shared nodes, base_part to traffic_part percentage is 50%, the number of available modules n=20. The results of RS and CO metrics for a k=6 fat-tree are shown in Figure 5.17. It worth-mentioning that the results of BFD_SINGLE_INSTANCE shown are with allocating percentages that are less than 1, which means not all requests have been allocated due to the high bandwidth requirement of the algorithm.



(a) Residual Resources

(b) Communication Overhead

Figure 5.17: RS and CO for The Probability of the Stateless Class, when k=6

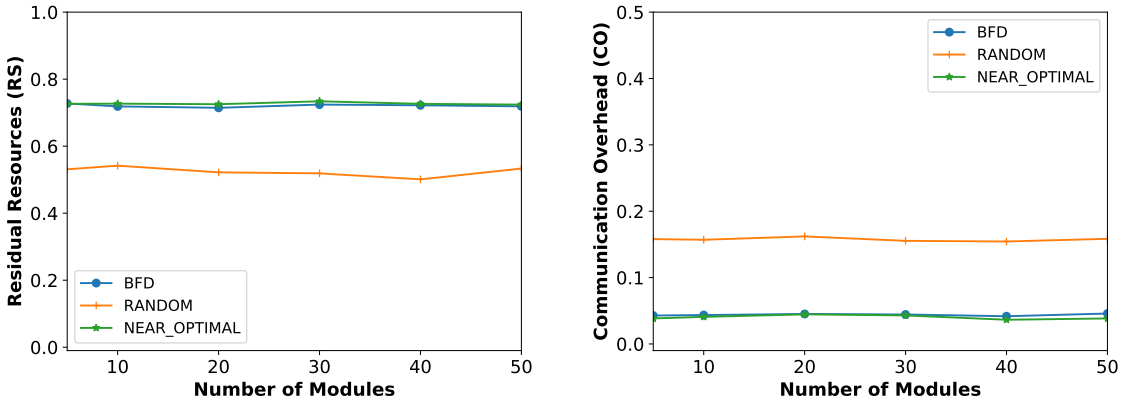
Figure 5.17a shows Residual Resources decreases with the increase of the probability of the stateless class p as the stateless class has complete duplicated elements compared to the stateful class where only monitoring elements are duplicated. While BFD_SINGLE_INSTANCE show steady performance as requests are allocated as one instance despite their type, furthermore, BFD_SINGLE_INSTANCE show more RS than all algorithms as it adopts one instance strategy while BFD show more RS than the RANDOM algorithm. Moreover, the results show NEAR_OPTIMAL algorithm spare more RS than BFD which is a result of

optimising each allocation to reduce residual resources will fill the fragments that may be resulted from BFD and increase overall residual resources.

Figure 5.17b shows the CO objective function after the allocation is complete. It shows as p increases, and the communication overhead decreases with the decrease of the stateful classes which impose no communication overhead to the system. Additionally, when p=1 and only stateless class is present, the communication overhead reaches zero. Furthermore, BFD_SINGLE_INSTANCE shows a steady case with both classes allocated in the same way. However, BFD_SINGLE_INSTANCE has less CO than other algorithms where it failed to allocate all requests. Furthermore, NEAR_OPTIMAL algorithm shows more CO than BFD which can be attributed to optimising for minimising residual resources will prefer stateless class over a stateful class where it requires more resources and resulting in more stateful class allocated in higher layers which increase CO.

5.5.6 Number of Modules

The next experiment shows *effect of Number of Modules* on the residual resources and communication overhead of the BFD, NEAR_OPTIMAL and RANDOM algorithms in case of modules size (required_resources) parameters are μ=60% and σ=10% of the maximum value of module size and traffic demand rate parameters are μ=60% and σ=10% of the maximum value of flow rate, tenant request rate r=4, communication overhead parameters are μ=50% and σ=10% of max value of flow rate between shared nodes, base_part to traffic_part percentage is 50%, and the probability of the stateless class p=50%. The number of modules tested is from the list $[10,\ 20,\ 30,\ 40,\ 50]$. The results of RS and CO metrics for a k=6 fat-tree are shown in Figure 5.18. The Figure shows the scale properties of the BFD and other algorithms where identical results for RS and CO are observed of modules numbers 10 to 50 for all algorithms.

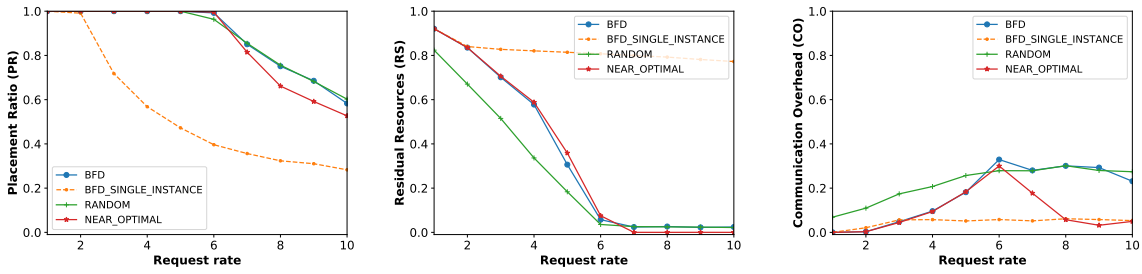


(a) Residual Resources (b) Communication Overhead

Figure 5.18: RS and CO for Number of Modules, when k=6

5.6 Scalability

The scalability of the BFD, BFD_SINGLE_INSTANCE, NEAR_OPTIMAL and RANDOM algorithms are shown by increasing the number of requests until the algorithms could not find a complete solution to the problem. Moreover, the experiment will test the performance results of the partial solution found. The experiment shows the *effect of request rate workload* on the performance metrics as the first experiment in Section Section 5.4 and in case of module size (required_resources) parameters are μ=70% and σ=10% of the maximum value of module size and the request rates are $\in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. The results of PR, RS and CO metrics for a k=6 fat-tree are shown Figure 5.19. It is worth mentioning that the experiment does not guarantee the system capacity constraint and that we omit the constraint programming where it could not obtain a partial solution.



(a) Placement Ratio (b) Residual Resources (c) Communication Overhead

Figure 5.19: PR, RS and CO for Request Rate, when k=6

Figure 5.19a show the PR objective function after the allocation is complete and, the results show PR=1 below r=6 where all requested resources have been allocated while show decreasing of PR beyond that. Moreover, results show that NEAR_OPTIMAL has less PR then BFD and RANDOM. This can be attributed that NEAR_OPTIMAL objective of filling locations with the most resources will prioritise stateless class over stateful class which results in allocating the more consuming resources class type and results in less RS to accommodate requests and subsequently less PR. Furthermore, BFD_SINGLE_INSTANCE has $PR < 1$ beyond r=2 which is a result of no available bandwidth to allocate requests with its steering strategy.

Figure 5.19b shows a decrease in residual resources below r=6 while showing steady results beyond that which can be attributed to the increase in the number of requests will result in a reduction in available resources until no more resources can be allocated. Furthermore, the results show that BFD_SINGLE_INSTANCE has more spare resources than BFD, where it could not allocate most of the requests as shown in Figure 5.19a. While BFD and NEAR_-OPTIMAL have more spare resources than the RANDOM algorithm as they utilise allocation to minimise resource consumption.

Figure 5.19c show the CO objective function after the allocation is complete and, the results show a linear increase of CO as more requests will introduce more communication overhead to the allocation while BFD_SINGLE_INSTANCE has less CO than other algorithms where it could not allocate most of the requests. Moreover, the results show that NEAR_OPTIMAL algorithm has a decreasing CO beyond r=6, which can be attributed to less stateful class functions being allocated, and less CO endured.

We conclude that increasing the request rate will result in algorithms failing to find a complete solution to the placement problem. However, the BFD algorithm has been able to scale and find a solution that will satisfy more computing resources requests than other algorithms including the near-optimal algorithms which will priorities one of the classes over the other and results in less CO but less requests satisfaction.

5.7 Summary

These chapter sections have presented the most important characteristics of the developed solutions to solve the placement of security function problems. It has used simulation results to compare the constraint programming, heuristic, meta-heuristic, near-optimal and LP implementation as well. Furthermore, it has explored the different characteristics of the proposed solutions against different factors, such as network size and the number of available security modules. The main findings of this chapter are the BFD algorithm based on sorting request on computing required resources of the requested modules has shown a balance between utilising computing and communication resources compared to other heuristic algorithms. It shows a near-optimal solution compared to the constraint programming solutions while solutions such as TABU meta-heuristic and near-optimal solutions have reached a slightly more utilisation to resources than the tested BFD. However, BFD has proved optimised time and success rate compared to other algorithms. When testing the legacy single instance allocation against BFD, it has shown better saving to computing resources that reached 50% while it showed more communication overhead up to 70% more than BFD. Moreover, an LP implementation that eliminated the communication overhead of the framework has been evaluated. The results show that an increasing amount of computing resources are required to accommodate requests compared to the CP model. Furthermore, the BFD version of the LP implementation is shown to have a near-optimal solution compared to the LP model solution. Furthermore, BFD shows scalability with increasing the request rate.

Chapter 6

Conclusion and Future Work

6.1 Overview

In this chapter, we summarise and conclude this work. The remainder of this chapter is structured as follows: Section 6.2 details the contributions made throughout this work. A discussion on directions for future work is presented in Section 6.3, including improvements and extensions to the current work. Finally, concluding remarks in Section 6.4.

6.2 Contribution Summary

This thesis has addressed the security function requirements and constraints of today's virtual network function orchestration and management frameworks. While a security function has been typically treated the same as any other network function, it poses unique requirements and constraints that have been identified and justified in this thesis. This work outlines the benefits and challenges of deploying network functions as VNFs in multi-tenant environments with an emphasis on security network functions. Furthermore, it analyses the previous approaches found in the literature to address similar problems and surveys their aims, methods and shortcoming in case of security functions.

This thesis has begun by detailing the benefits that NFV and SDN bring to network func-

tion deployment (e.g., software-based network functions, efficient resource provisioning, the flexibility of placement, etc.) compared to their counterpart hardware middleboxes in legacy networks. Then, it illustrates the rising of customised security services in a multi-tenant virtual environment which motivated the aims of future customised security VNFs provided as services for tenants.

While there is considerable work that has addressed the virtual network function management problem, this thesis showed that security functions enforce placement constraints that are not presenting in other types of network functions in a multi-tenant environment. For example, the different granularity of the traffic required by the security functions classes will constraint the placement of the function to locations where it can be satisfied. Moreover, the complexity of the security functionality, the stateful class in particular, limits function sharing among multiple tenants. Besides, identifying north-south traffic as the traffic direction required for security classes in contrast to east-west traffic promoted as the designated traffic direction of network functions by previous approaches. Moreover, some constraints will increase the required resources for the deployment process. For example, the non-sharing policy will increase the number of instances while the granularity of security functions will limit function placement location or force traffic steering. Furthermore, the limitations of current research and solutions to address these constraints and requirements have been discussed.

The work conducted in this thesis focused on the design of a security placement framework that utilises the SDN and VNF technologies to allow system operators to provide customised security services to their tenants. Increasing their capability for dynamic security solutions that can elastically face security threats at the run time. The placement framework is tailored to security functions, and aimed at reducing complexity and saving computing and communication resources. It implements a placement strategy that satisfies the unique constraints of the security functions and reduces the resource overhead due to the constraints enforces by the deployment. The framework provides customised security services to meet the diversity of users in multi-tenant environments. Furthermore, the placement problem of the

framework has been analysed and modelled mathematically.

Heuristic, meta-heuristic (tabu), near-optimal and constraint programming methods have been evaluated as solutions to the placement problem. The results show that the BFD heuristic algorithm shows up to 70% less CO and up to 50% more than in RS compared to RANDOM and FFD heuristic solutions. It also shows that the Tabu algorithms that start with a BFD solution can not achieve better utilisation to resources while tabu that starts with a random solution can achieve slightly higher utilisation. Moreover, the near-optimal solution was able to spare up to 5% more residual resources than BFD and has the same residual resources as the constraint programming solution while communication overhead of the BFD algorithm is almost overlapping with the near-optimal solution. However, constraint programming shows less CO than BFD that reach up 10% while it shows the same as BFD at high workload. Furthermore, the single instance strategy has shown 50% more residual resources than BFD while BFD has reached 50% less CO. The results also show that the BFD algorithm outperforms tabu and near-optimal solutions in execution time and scalability while they slightly outperform BFD in optimality.

6.3 Future Work

6.3.1 Supporting Placement of Security VNF Chains

To simplify the problem, the design of the placement framework assumes a user can request multiple security functions but it does not support a specific order of the functions of the request. However, security policy usually involves processing traffic in a specific order such as a firewall then an IDS which has been known as a chain of requests. Supporting chains of requests will add a chain location optimisation as an objective to the problem which converts it to another complex problem where our objective has been saving resources while satisfying the identified constraints of security modules placement. Moreover, it will require changes to the placement algorithm.

6.3.2 Dynamic Placement

While the current framework supports the static version (initial placement) of the placement problem, in a real situation, a migration algorithm is to be implemented as well to handle dynamic (run-time) problems such as overloaded servers, not enough space of scaling up, etc. These problems usually arise from the dynamic nature of the environment where changes can occur at all levels such as VM migration or changes in traffic, plus the dynamic nature of security where a user comes under attack will require changes to their services at run-time. Besides, a migration algorithm is to determine a policy when migrating or rearranging VNFs is required. Typically, it will take decisions such as which function to migrate and where to. Moreover, migrating algorithms will also be designed with the objective of the initial placement, in our case, save resources and reduce the number of migrations as well as migrating VNF can affect QoS due to the stop/start operations.

Besides, migration has many types such as stateless, quick, and live migration which identifies how it moves the VNF and internal state reservation policy. The type of migration selected would highly affect a security function where the stateful class, for example, depends on previous data that was stored for comparison to the new traffic pattern. Therefore, it would be interesting to explore how migration types can support the stateful class while keeping high QoS to the highly sensitive security services which is a non-trivial process.

6.3.3 Exploring Real Data Center Architectures

This work presented an implementation of the proposed placement framework on the fat-tree architecture as a widely used architecture in data centers. However, real data center deployments include modifications to enhance certain characteristic. A future direction may include testing the proposed solution with other typologies such as the spine plane architecture of Facebook [166] or the Jupiter topology of Google [167].

6.3.4 QoS Constraints

Security services offered by cloud providers allow different service types to the same security module such as different throughput based on CPU cores deployed. The proposed framework assumes that throughput is granted by the initial resources required per module. QoS parameters such as throughput requested by a user add a further dimension to the placement problem that can help save resources and offer more customised services to fit users' needs. It also adds more complexity to the placement problem represented as more constraints are added to the placement.

6.4 Summary and Concluding Remarks

The growth in numbers of ICT companies moving to the cloud comes with more network services offered by cloud services providers to users. In addition to the rising in security threats such as DDoS and ransom attacks that target all size companies, research is focusing on improving the security network function softwarisation and virtualisation. While the dynamic control of service orchestration of security functionality is a complex process that is based on user requirements and QoS with the objective to increase profit and reduce expenditure.

come form section The work presented in this thesis proposes a placement framework that addresses these constraints and a placement strategy to minimise the resource overhead due to the function deployment. It exploits collocations with network switches for VNF deployment which allows on-path deployment for the security function and reduces traffic steering overhead. Moreover, it adopts distributed deployment of the stateful class which reduces the communication overhead of the deployment compared to the single instance strategy. The framework placement problem has been addressed mathematically with the objective to minimise both computing and communication resource consumption of the infrastructure with a solution provided in Constraint Programming for optimal placement. This thesis introduces a heuristic solution to the problem as a time-optimised solution that will balance the trade of computing and communication resource consumption in the placement process. The

proposed solution has proven scalable and near-optimal compared to previously proposed meta-heuristic and near-optimal subset-sum solutions.

This work intends to prove the hypothesis that security functions pose characteristics that are not present in case of other network functions. These characteristics enforce constraints to the orchestration process and increase resource overhead of the placement. Through this work, operators can orchestrate security network function as VNFs to provide customised security services to users. This work has shown that the placement of security VNFs can satisfy the security constraints and maintain efficient management of the network-wide resources.

Publication

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

- Ali, Abeer, Christos Anagnostopoulos, and Dimitrios P. Pezaros. ”In-Network Placement of Security VNFs in Multi-Tenant Data Centers.” In 2020 IEEE Symposium on Computers and Communications (ISCC), 2020.
- Ali, Abeer, Christos Anagnostopoulos, and Dimitrios P. Pezaros. ”On the Optimality of Virtualized Security Function Placement in Multi-Tenant Data Centers.” In 2018 IEEE International Conference on Communications (ICC), pp. 1-6. IEEE, 2018.
- Ali, Abeer, 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), pp. 1-5. IEEE, 2017.
- Ali, Abeer, Richard Cziva, Simon Jouet, and Dimitrios P. Pezaros. ”SDNFV-based DDoS detection and remediation in multi-tenant, virtualised infrastructures.” In Guide to Security in SDN and NFV, pp. 171-196. Springer, Cham, 2017.

Bibliography

[1] F. G. Perraudeau, "Middleboxes as a Cloud Service," Ph.D. dissertation, UC Berkeley, 2018.

[2] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '12. New York, NY, USA: ACM, 2012, pp. 13–24. [Online]. Available: http://doi.acm.org/10.1145/2342356.2342359

[3] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, V. Sekar, and A. Akella, "Stratos: A network-aware orchestration layer for virtual middleboxes in clouds," *arXiv preprint arXiv:1305.0209*, 2013.

[4] S. Rajagopalan, D. Williams, and H. Jamjoom, "Pico replication: A high availability framework for middleboxes," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 1:1–1:15. [Online]. Available: http://doi.acm.org/10.1145/2523616.2523635

[5] L. Gyarmati and T. A. Trinh, *Energy Efficiency of Data Centers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 229–244. [Online]. Available: https://doi.org/10.1007/978-3-642-22179-8_12

[6] S. U. Khan and A. Y. Zomaya, *Handbook on data centers*. Springer, 2015.

[7] C. Basile, C. Pitscheider, F. Risso, F. Valenza, and M. Vallini, "Towards the dynamic provision of virtualized security services," in *Cyber Security and Privacy Forum*. Springer, 2015, pp. 65–76.

[8] S. Shin, H. Wang, and G. Gu, "A first step toward network security virtualization: from concept to prototype," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 10, pp. 2236–2249, 2015.

[9] (2017) Silver peak. [Online]. Available: https://www.silver-peak.com/

[10] (2019) Riverbed. [Online]. Available: https://www.riverbed.com/gb/

[11] Cisco. Cisco. [Online]. Available: https://www.cisco.com/

[12] sophos. Sophos. [Online]. Available: https://home.sophos.com/en-us.aspx

[13] (2017) Snort intrusion detection system. [Online]. Available: https://www.snort.org/

[14] (2017) The suricata open source ids, ips, and nsm. [Online]. Available: https://suricata-ids.org/

[15] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: methods, systems and tools," *Ieee communications surveys & tutorials*, vol. 16, no. 1, pp. 303–336, 2014.

[16] C. Douligeris and A. Mitrokotsa, "Ddos attacks and defense mechanisms: classification and state-of-the-art," *Computer Networks*, vol. 44, no. 5, pp. 643–666, 2004.

[17] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[18] B. B. Gupta and O. P. Badve, "Taxonomy of dos and ddos attacks and desirable defense mechanism in a cloud computing environment," *Neural Computing and Applications*, pp. 1–28, 2016. [Online]. Available: http://dx.doi.org/10.1007/s00521-016-2317-5

[19] S. M. Specht and R. B. Lee, "Distributed denial of service: Taxonomies of attacks, tools, and countermeasures," in *ISCA PDCS*, 2004, pp. 543–550.

[20] O. Osanaiye, K.-K. R. Choo, and M. Dlodlo, "Distributed denial of service (ddos) resilience in cloud: review and conceptual cloud ddos mitigation framework," *Journal of Network and Computer Applications*, vol. 67, pp. 147–165, 2016.

[21] (2018) DDoS Incident Report - GitHub Engineering. [Online]. Available: https://githubengineering.com/ddos-incident-report/

[22] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 602–622, Firstquarter 2016.

[23] F. Wong and C. Xiang Tan, "A survey of trends in massive ddos attacks and cloud-based mitigations," *International Journal of Network Security & Its Applications*, vol. 6, pp. 57–71, 05 2014.

[24] Akamai, "state of the internet - security : DDoS and Application Attacks Report," Akamai, Tech. Rep., 2019. [Online]. Available: https://www.akamai.com/us/en/resources/our-thinking/state-of-the-internet-report/global-state-of-the-internet-security-ddos-attack-reports.jsp

[25] G. Somani, M. S. Gaur, D. Sanghi, M. Conti, and R. Buyya, "Ddos attacks in cloud computing: Issues, taxonomy, and future directions," *Computer Communications*, vol. 107, pp. 30 – 48, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366417303791

[26] G. Somani, M. S. Gaur, and D. Sanghi, "Ddos/edos attack in cloud: Affecting everyone out there!" in *Proceedings of the 8th International Conference on Security of Information and Networks*, ser. SIN '15. New York, NY, USA: ACM, 2015, pp. 169–176. [Online]. Available: http://doi.acm.org/10.1145/2799979.2800005

[27] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in Cloud," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 42–57, jan 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804512001178

[28] M. G. Gouda and A. X. Liu, "A model of stateful firewalls and its properties," in *2005 International Conference on Dependable Systems and Networks (DSN'05)*, June 2005, pp. 128–137.

[29] K. A. Scarfone and P. M. Mell, "SP 800-94. Guide to Intrusion Detection and Prevention Systems (IDPS)," Gaithersburg, MD, United States, Tech. Rep., 2007.

[30] H. T. Elshoush and I. M. Osman, "Alert correlation in collaborative intelligent intrusion detection systems—A survey," *Applied Soft Computing*, vol. 11, no. 7, pp. 4349–4365, oct 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S156849461000311X

[31] H.-J. Liao, C.-H. Richard Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, jan 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804512001944

[32] H. Debar, M. Dacier, and A. Wespi, "A revised taxonomy for intrusion-detection systems," *Annales Des Télécommunications*, vol. 55, no. 7-8, pp. 361–378, 2000. [Online]. Available: https://link.springer.com/article/10.1007/BF02994844

[33] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An Overview of IP Flow-Based Intrusion Detection," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 3, pp. 343–356, 2010. [Online]. Available: http://ieeexplore.ieee.org/document/5455789/

[34] A. Patel, M. Taghavi, K. Bakhtiyari, and J. Celestino Júnior, "An intrusion detection and prevention system in cloud computing: A systematic review," *Journal of Network*

and Computer Applications, vol. 36, no. 1, pp. 25–41, jan 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S108480451200183X

[35] G. Münz and G. Carle, "Real-time analysis of flow data for network attack detection," in *10th IFIP/IEEE International Symposium on Integrated Network Management 2007, IM '07*. IEEE, may 2007, pp. 100–108. [Online]. Available: http://ieeexplore.ieee.org/document/4258526/

[36] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers & Security*, vol. 28, no. 1-2, pp. 18–28, feb 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404808000692

[37] J. J. Davis and A. J. Clark, "Data preprocessing for anomaly based network intrusion detection: A review," *Computers & Security*, vol. 30, no. 6-7, pp. 353–375, sep 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404811000691

[38] A. A. Ghorbani, W. Lu, and M. Tavallaee, *Detection Approaches*. Boston, MA: Springer US, 2010, pp. 27–53. [Online]. Available: https://doi.org/10.1007/978-0-387-88771-5_2

[39] H. Han, X. L. Lu, J. Lu, C. Bo, and R. L. Yong, "Data mining aided signature discovery in network-based intrusion detection system," *ACM SIGOPS Operating Systems Review*, vol. 36, no. 4, pp. 7–13, oct 2002. [Online]. Available: http://portal.acm.org/citation.cfm?doid=583800.583801

[40] (2017) The bro network security monitor. [Online]. Available: https://www.bro.org/

[41] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer Networks*, vol. 51, no. 12, pp. 3448–3470, aug 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138912860700062X

[42] S. Smaha, "Haystack: an intrusion detection system," in *[Proceedings 1988] Fourth Aerospace Computer Security Applications*. IEEE Comput. Soc. Press, 1998, pp. 37–44. [Online]. Available: http://ieeexplore.ieee.org/document/113412/

[43] C. Krügel, T. Toth, and E. Kirda, "Service specific anomaly detection for network intrusion detection," in *Proceedings of the 2002 ACM symposium on Applied computing - SAC '02*. New York, New York, USA: ACM Press, 2002, p. 201. [Online]. Available: http://portal.acm.org/citation.cfm?doid=508791.508835

[44] Wenke Lee and Dong Xiang, "Information-theoretic measures for anomaly detection," in *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*. IEEE Comput. Soc, 2001, pp. 130–143. [Online]. Available: http://ieeexplore.ieee.org/document/924294/

[45] M. N. Ismail, A. Aborujilah, S. Musa, and A. Shahzad, "Detecting flooding based DoS attack in cloud computing environment using covariance matrix approach," in *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication - ICUIMC '13*. New York, New York, USA: ACM Press, 2013, pp. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2448556.2448592

[46] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava, "A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection," in *Proceedings of the 2003 SIAM International Conference on Data Mining*. Philadelphia, PA: Society for Industrial and Applied Mathematics, may 2003, pp. 25–36. [Online]. Available: https://epubs.siam.org/doi/10.1137/1.9781611972733.3

[47] B. Subba, S. Biswas, and S. Karmakar, "A Neural Network based system for Intrusion Detection and attack classification," in *2016 Twenty Second National Conference on Communication (NCC)*. IEEE, mar 2016, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/7561088/

[48] W.-T. Wong and S.-H. Hsu, "Application of SVM and ANN for image retrieval," *European Journal of Operational Research*, vol. 173, no. 3, pp. 938–950, sep 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221705006922

[49] S. M. Bridges, R. B. Vaughn, and Others, "Fuzzy data mining and genetic algorithms applied to intrusion detection," in *Proceedings of 12th Annual Canadian Information Technology Security Symposium*, 2000, pp. 109–122.

[50] Y. Dhanalakshmi and I. Ramesh Babu, "Intrusion Detection Using Data Mining Along Fuzzy Logic and Genetic Algorithms," *International Journal of Computer Science and Network Security*, vol. 8, 2008.

[51] D.-Y. Yeung and Y. Ding, "Host-based intrusion detection using dynamic and static behavioral models," *Pattern Recognition*, vol. 36, no. 1, pp. 229–243, jan 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320302000262

[52] K. Wang and S. J. Stolfo, "Anomalous Payload-Based Network Intrusion Detection," in *Recent Advances in Intrusion Detection*, E. Jonsson, A. Valdes, and M. Almgren, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 203–222.

[53] H. Beitollahi and G. Deconinck, "ConnectionScore: a statistical technique to resist application-layer DDoS attacks," *Journal of Ambient Intelligence and Humanized Computing*, vol. 5, no. 3, pp. 425–442, jun 2014. [Online]. Available: http://link.springer.com/10.1007/s12652-013-0196-5

[54] F. Valeur, D. Mutz, and G. Vigna, "A learning-based approach to the detection of sql attacks," in *Proceedings of the Second International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. DIMVA'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 123–140. [Online]. Available: http://dx.doi.org/10.1007/11506881_8

[55] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 27–38, Aug. 2013.

[56] Xin Li and Chen Qian, "A survey of network function placement," in *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, jan 2016, pp. 948–953. [Online]. Available: http://ieeexplore.ieee.org/document/7444915/

[57] Cisco. Installing the IDS Appliance - Cisco. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/security/ips/4-0/installation/guide/

[58] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, "Ddos attack protection in the era of cloud computing and software-defined networking," in *2014 IEEE 22nd International Conference on Network Protocols*, Oct 2014, pp. 624–629.

[59] D. A. Joseph, A. Tavakoli, and I. Stoica, "A policy-aware switching layer for data centers," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 51–62, Aug. 2008. [Online]. Available: http://doi.acm.org/10.1145/1402946.1402966

[60] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan *et al.*, "Steering: A software-defined networking for inline service chaining," in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*. IEEE, 2013, pp. 1–10.

[61] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 459–473. [Online]. Available: http://dl.acm.org/citation.cfm?id=2616448.2616491

[62] R. Cziva, S. Jouet, and D. P. Pezaros, "Gnfc: Towards network function cloudifi-

cation," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, Nov 2015, pp. 142–148.

[63] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[64] A. Ali, R. Cziva, S. Jouët, and D. P. Pezaros, *SDNFV-Based DDoS Detection and Remediation in Multi-tenant, Virtualised Infrastructures*. Cham: Springer International Publishing, 2017, pp. 171–196. [Online]. Available: https://doi.org/10.1007/978-3-319-64653-4_7

[65] FortiGate. FortiGate Next Generation Firewall Virtual Appliance (NGFW). [Online]. Available: https://www.fortinet.com/products/private-cloud-security/fortigate-virtual-appliances.html

[66] Juniper. Juniper vSRX Virtual Firewall. [Online]. Available: https://www.juniper.net/us/en/products-services/security/srx-series/vsrx/

[67] Cisco. Cisco Adaptive Security Virtual Appliance (ASAv). [Online]. Available: https://www.cisco.com/c/en/us/products/security/virtual-adaptive-security-appliance-firewall/index.html

[68] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721672

[69] "Cisco Virtualized Multi-Tenant Data Center Design Guide Version 2.2 - Architecture Overview [Data Center Designs: Virtualization] - Cisco." [Online]. Available: https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data{_}Center/VMDC/2-2/design{_}guide/vmdcDesign22/VMDC{_}2-2{_}DG{_}1.html

[70] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, Firstquarter 2016.

[71] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Communications Surveys Tutorials*, pp. 1–1, 2018.

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

[73] R. Cziva and D. P. Pezaros, "Container network functions: Bringing nfv to the network edge," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 24–31, June 2017.

[74] H. Sadok, M. Campista, and L. Costa, "Improving software middleboxes and datacenter task schedulers," 09 2019, pp. 137–144.

[75] C. Yoon, T. Park, S. Lee, H. Kang, S. Shin, and Z. Zhang, "Enabling security functions with sdn: A feasibility study," *Computer Networks*, vol. 85, pp. 19 – 35, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128615001619

[76] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software defined networking: State of the art and research challenges," *Computer Networks*, vol. 72, pp. 74 – 98, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128614002588

[77] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 1955–1980, Fourthquarter 2014.

[78] M. Xia, M. Shirazipour, Y. Zhang, H. Green, and A. Takacs, "Network function placement for nfv chaining in packet/optical data centers," *Journal of Lightwave Technology*, vol. 33, no. 8, pp. 1565–1570, April 2015.

[79] M. M. Tajiki, S. Salsano, L. Chiaraviglio, M. Shojafar, and B. Akbari, "Joint energy efficient and qos-aware path allocation and vnf placement for service function chaining," *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, pp. 374–388, 2019.

[80] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek, "The click modular router," *SIGOPS Oper. Syst. Rev.*, vol. 33, no. 5, pp. 217–231, Dec. 1999. [Online]. Available: http://doi.acm.org/10.1145/319344.319166

[81] B. Anwer, T. Benson, N. Feamster, and D. Levin, "Programming slick network functions," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, ser. SOSR '15. New York, NY, USA: ACM, 2015, pp. 14:1–14:13. [Online]. Available: http://doi.acm.org/10.1145/2774993.2774998

[82] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "Opennf: Enabling innovation in network function control," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 163–174, Aug. 2014. [Online]. Available: http://doi.acm.org/10.1145/2740070.2626313

[83] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 533–546. [Online]. Available: http://dl.acm.org/citation.cfm?id=2616448.2616497

[84] (2019) Opnfv. [Online]. Available: https://www.opnfv.org/

[85] (2019) Open source mano. [Online]. Available: https://osm.etsi.org/

[86] N. ETSI, "Gs nfv-man 001 v1. 1.1 network function virtualisation (nfv); management and orchestration," 2014.

[87] K. Giotis, Y. Kryftis, and V. Maglaris, "Policy-based orchestration of nfv services

in software-defined networks," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, April 2015, pp. 1–5.

[88] M. C. Luizelli, D. Raz, Y. Sa'ar, and J. Yallouz, "The actual cost of software switching for nfv chaining," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, May 2017, pp. 335–343.

[89] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862 – 876, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128609003387

[90] M. Gao, B. Addis, M. Bouet, and S. Secci, "Optimal orchestration of virtual network functions," *Computer Networks*, vol. 142, pp. 108 – 127, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128618303578

[91] D. Dwiardhika and T. Tachibana, "Virtual network embedding based on security level with vnf placement," *Security and Communication Networks*, vol. 2019, 2019.

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

[93] F. Carpio, S. Dhahri, and A. Jukan, "Vnf placement with replication for loac balancing in nfv networks," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–6.

[94] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and S. Davy, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, April 2015, pp. 1–9.

[95] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM)*, ser. CNSM '15. Washington,

DC, USA: IEEE Computer Society, 2015, pp. 50–56. [Online]. Available: http://dx.doi.org/10.1109/CNSM.2015.7367338

[96] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, Oct 2015, pp. 171–177.

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

[98] L. Yala, P. A. Frangoudis, and A. Ksentini, "Latency and availability driven VNF placement in a MEC-NFV environment," in *GLOBECOM 2018, IEEE Global Communications Conference, 9-13 December 2018, Abu Dhabi, UAE*, Abu Dhabi, UNITED ARAB EMIRATES, 12 2018. [Online]. Available: http://www.eurecom.fr/publication/5649

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

[100] L. Cui, R. Cziva, F. P. Tso, and D. P. Pezaros, "Synergistic policy and virtual machine consolidation in cloud data centers," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.

[101] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, "E2: A framework for nfv applications," in *Proceedings of the 25th Symposium on Operating Systems Principles*, ser. SOSP '15. New York, NY, USA: ACM, 2015, pp. 121–136. [Online]. Available: http://doi.acm.org/10.1145/2815400.2815423

[102] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, and H. A. Chan, "Optimal virtual network function placement in multi-cloud service function chaining

architecture," *Computer Communications*, vol. 102, pp. 1 – 16, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366417301901

[103] A. Grochowski, "Virtual machine placement strategies for virtual network functions," NCTA-2016 Spring Technical Forum Proceedings, Tech. Rep., 2016. [Online]. Available: https://www.nctatechnicalpapers.com/Paper/2016/2016-virtual-machine-placement-strategies-for-virtual-network

[104] M. Bouet, J. Leguay, and V. Conan, "Cost-based placement of vdpi functions in nfv infrastructures," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, April 2015, pp. 1–9.

[105] S. Demirci and S. Sagiroglu, "Optimal placement of virtual network functions in software defined networks: A survey," *Journal of Network and Computer Applications*, vol. 147, p. 102424, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804519302760

[106] M. Yoshida, W. Shen, T. Kawabata, K. Minato, and W. Imajuku, "Morsa: A multi-objective resource scheduling algorithm for nfv infrastructure," in *The 16th Asia-Pacific Network Operations and Management Symposium*, Sep. 2014, pp. 1–6.

[107] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. 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)*, May 2015, pp. 98–106.

[108] T. Wen, H. Yu, G. Sun, and L. Liu, "Network function consolidation in service function chaining orchestration," in *2016 IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–6.

[109] T. Kuo, B. Liou, K. C. Lin, and M. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1562–1576, Aug 2018.

[110] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, April 2015, pp. 1346–1354.

[111] C. Pham, N. H. Tran, S. Ren, W. Saad, and C. S. Hong, "Traffic-aware and energy-efficient vnf placement for service chaining: Joint sampling and matching approach," *IEEE Transactions on Services Computing*, pp. 1–1, 2018.

[112] C. Hsieh, J. Chang, C. Chen, and S. Lu, "Network-aware service function chaining placement in a data center," in *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Oct 2016, pp. 1–6.

[113] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, "An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2008–2025, Aug 2017.

[114] L. Qu, C. Assi, and K. Shaban, "Delay-aware scheduling and resource optimization with network function virtualization," *IEEE Transactions on Communications*, vol. 64, no. 9, pp. 3746–3758, Sep. 2016.

[115] D. Qi, S. Shen, and G. Wang, "Towards an efficient vnf placement in network function virtualization," *Computer Communications*, vol. 138, pp. 81 – 89, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366418308247

[116] J. Pei, P. Hong, M. Pan, J. Liu, and J. Zhou, "Optimal vnf placement via deep reinforcement learning in sdn/nfv-enabled networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 263–278, 2020.

[117] A. Bremler-Barr, Y. Harchol, and D. Hay, "Openbox: A software-defined framework for developing, deploying, and managing network functions," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16.

New York, NY, USA: ACM, 2016, pp. 511–524. [Online]. Available: http://doi.acm.org/10.1145/2934872.2934875

[118] L. Foundation, "Linux foundation open vswitch," https://LinuxFoundationOpenvSwitch, March 2017, (Accessed on 28/03/2017).

[119] X. Zhang, X. Wang, C. Nguyen, J. Wang, Z. Qian, and S. Lu, "A virtual middleboxes network placement algorithm in multi-tenant datacenter networks," in *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*, Dec 2017, pp. 445–452.

[120] K. Cabaj, J. Wytrebowicz, S. Kuklinski, P. Radziszewski, and K. T. Dinh, "Sdn architecture impact on network security," in *FedCSIS position papers*, 2014, pp. 143–148.

[121] H. Hamad and M. Al-hoby, "Article: Managing intrusion detection as a service in cloud networks," *International Journal of Computer Applications*, vol. 41, no. 1, pp. 35–40, March 2012.

[122] S. Roschke, F. Cheng, and C. Meinel, "Intrusion detection in the cloud," in *2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*, Dec 2009, pp. 729–734.

[123] X. He, T. Guo, E. M. Nahum, and P. Shenoy, "Placement strategies for virtualized network functions in a nfaas cloud," in *2016 Fourth IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, Oct 2016, pp. 48–53.

[124] A. Shameli-Sendi, Y. Jarraya, M. Pourzandi, and M. Cheriet, "Efficient provisioning of security service function chaining using network security defense patterns," *IEEE Transactions on Services Computing*, vol. 12, no. 4, pp. 534–549, 2019.

[125] M. Ghaznavi, N. Shahriar, S. Kamali, R. Ahmed, and R. Boutaba, "Distributed service function chaining," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2479–2489, Nov 2017.

[126] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '04. New York, NY, USA: ACM, 2004, pp. 219–230. [Online]. Available: http://doi.acm.org/10.1145/1015467.1015492

[127] T. Huang, H. Sethu, and N. Kandasamy, "A new approach to dimensionality reduction for anomaly detection in data traffic," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 651–665, Sep. 2016.

[128] L. Huang, X. Nguyen, M. Garofalakis, M. I. Jordan, A. Joseph, and N. Taft, "In-network pca and anomaly detection," in *Proceedings of the 19th International Conference on Neural Information Processing Systems*, ser. NIPS'06. Cambridge, MA, USA: MIT Press, 2006, pp. 617–624. [Online]. Available: http://dl.acm.org/citation.cfm?id=2976456.2976534

[129] R. Keralapura, G. Cormode, and J. Ramamirtham, "Communication-efficient distributed monitoring of thresholded counts," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '06. New York, NY, USA: ACM, 2006, pp. 289–300. [Online]. Available: http://doi.acm.org/10.1145/1142473.1142507

[130] J. Dromard, G. Roudiere, and P. Owezarski, "Online and scalable unsupervised network anomaly detection method," *IEEE Trans. on Netw. and Serv. Manag.*, vol. 14, no. 1, pp. 34–47, Mar. 2017. [Online]. Available: https://doi.org/10.1109/TNSM.2016.2627340

[131] J. B. Predd, S. B. Kulkarni, and H. V. Poor, "Distributed learning in wireless sensor networks," *IEEE Signal Processing Magazine*, vol. 23, no. 4, pp. 56–69, July 2006.

[132] N. Terzenidis, M. Moralis-Pegios, G. Mourgias-Alexandris, T. Alexoudi, K. Vyrsokinos, and N. Pleros, "High-port and low-latency optical switches for disaggregated data

centers: The hipoλaos switch architecture," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 10, no. 7, pp. 102–116, July 2018.

[133] C. V. Networking, "Cisco global cloud index: Forecast and methodology 2016–2021," *White paper*, 2018.

[134] (2019) Securing 'East-West' Traffic in the Cloud. [Online]. Available: https://www.bankinfosecurity.com/interviews/interview-vmware-coo-raghu-raghuram-i-4151

[135] M. G. Gouda and A. X. Liu, "A model of stateful firewalls and its properties," in *2005 International Conference on Dependable Systems and Networks (DSN'05)*, June 2005, pp. 128–137.

[136] T. AbuHmed, A. Mohaisen, and D. Nyang, "A survey on deep packet inspection for intrusion detection systems," *CoRR*, vol. abs/0803.0037, 2008. [Online]. Available: http://arxiv.org/abs/0803.0037

[137] A. G. Tartakovsky, B. L. Rozovskii, R. B. Blazek, and H. Kim, "A novel approach to detection of intrusions in computer networks via adaptive sequential and batch-sequential change-point detection methods," *IEEE Transactions on Signal Processing*, vol. 54, no. 9, pp. 3372–3382, Sep. 2006.

[138] P. Bereziński, B. Jasiul, and M. Szpyrka, "An entropy-based network anomaly detection method," *Entropy*, vol. 17, no. 4, pp. 2367–2408, 2015. [Online]. Available: http://www.mdpi.com/1099-4300/17/4/2367

[139] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the dos and ddos problems," *ACM Comput. Surv.*, vol. 39, no. 1, Apr. 2007. [Online]. Available: http://doi.acm.org/10.1145/1216370.1216373

[140] I. Karim, Q.-T. Vien, T. A. Le, and G. Mapp, "A comparative experimental design and performance analysis of snort-based intrusion detection system in practical computer networks," *Computers*, vol. 6, no. 1, 2017. [Online]. Available: http://www.mdpi.com/2073-431X/6/1/6

[141] J. S. White, T. Fitzsimmons, and J. N. Matthews, "Quantitative analysis of intrusion detection systems: Snort and suricata," in *Cyber Sensing 2013*, vol. 8757. International Society for Optics and Photonics, 2013, p. 875704.

[142] K. Salah and A. Kahtani, "Performance evaluation comparison of snort nids under linux and windows server," *Journal of Network and Computer Applications*, vol. 33, no. 1, pp. 6 – 15, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804509001040

[143] W. Bul'ajoul, A. James, and M. Pannu, "Improving network intrusion detection system performance through quality of service configuration and parallel technology," *Journal of Computer and System Sciences*, vol. 81, no. 6, pp. 981 – 999, 2015, special Issue on Optimisation, Security, Privacy and Trust in E-business Systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022000014001767

[144] M. Fisk and G. Varghese, "Applying fast string matching to intrusion detection," Los Alamos National Lab., NM (US), Tech. Rep., 2001.

[145] C. Sanders and J. Smith, *Applied Network Security Monitoring: Collection, Detection, and Analysis*, 1st ed. Syngress Publishing, 2013.

[146] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation algorithms for virtualized service hosting platforms," *Journal of Parallel and Distributed Computing*, vol. 70, no. 9, pp. 962 – 974, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731510000997

[147] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi, "Dynamic placement for clustered web applications," in *Proceedings of the 15th International Conference on World Wide Web*, ser. WWW '06. New York, NY, USA: ACM, 2006, pp. 595–604. [Online]. Available: http://doi.acm.org/10.1145/1135777.1135865

[148] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning

system for the cloud," in *2011 31st International Conference on Distributed Computing Systems*, June 2011, pp. 559–570.

[149] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755 – 768, 2012, special Section: Energy efficiency in large-scale distributed systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X11000689

[150] D. S. Johnson, *Bin Packing*. New York, NY: Springer New York, 2016.

[151] M. Iori, S. Martello, and M. Monaci, *Metaheuristic Algorithms for the Strip Packing Problem*. Boston, MA: Springer US, 2003, pp. 159–179. [Online]. Available: https://doi.org/10.1007/978-1-4613-0233-9_7

[152] K. Fleszar and C. Charalambous, "Average-weight-controlled bin-oriented heuristics for the one-dimensional bin-packing problem," *European Journal of Operational Research*, vol. 210, no. 2, 2011.

[153] J. L. Viegas, S. M. Vieira, E. M. P. Henriques, and J. M. C. Sousa, "A tabu search algorithm for the 3d bin packing problem in the steel industry," in *CONTROLO'2014 – Proceedings of the 11th Portuguese Conference on Automatic Control*, A. P. Moreira, A. Matos, and G. Veiga, Eds. Cham: Springer International Publishing, 2015, pp. 355–364.

[154] S. Ahvar, M. M. Mirzaei, J. Leguay, E. Ahvar, A. M. Medhat, N. Crespi, and R. Glitho, "Set: a simple and effective technique to improve cost efficiency of vnf placement and chaining algorithms for network service provisioning," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, June 2018, pp. 293–297.

[155] A. Leivadeas, G. Kesidis, M. Ibnkahla, and I. Lambadaris, "Vnf placement optimization at the edge and cloud," *Future Internet*, vol. 11, no. 3, 2019. [Online]. Available: https://www.mdpi.com/1999-5903/11/3/69

[156] M. Yue, "A simple proof of the inequality FFD (L) $\leq$ 11/9 opt (L) + 1, $\forall$L for the FFD bin-packing algorithm," *Acta Mathematicae Applicatae Sinica*, vol. 7, no. 4, pp. 321–331, 1991.

[157] M. R. Chowdhury, M. R. Mahmud, and R. M. Rahman, "Implementation and performance analysis of various vm placement strategies in cloudsim," *Journal of Cloud Computing*, vol. 4, no. 1, p. 20, Nov 2015. [Online]. Available: https://doi.org/10.1186/s13677-015-0045-5

[158] J. Huang, K. Wu, and M. Moh, "Dynamic virtual machine migration algorithms using enhanced energy consumption model for green cloud data centers," in *2014 International Conference on High Performance Computing Simulation (HPCS)*, July 2014, pp. 902–910.

[159] M. Haouari and M. Serairi, "Heuristics for the variable sized bin-packing problem," *Comput. Oper. Res.*, vol. 36, no. 10, pp. 2877–2884, Oct. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.cor.2008.12.016

[160] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *2010 Proceedings IEEE INFOCOM*, March 2010, pp. 1–9.

[161] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1402958.1402967

[162] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," Internet Requests for Comments, Internet Engineering Task Force, RFC 2992, November 2000.

[163] J. Lee, J. Tourrilhes, P. Sharma, and S. Banerjee, "No more middlebox: Integrate processing into network," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 459–460, Aug. 2010. [Online]. Available: http://doi.acm.org/10.1145/1851275.1851262

[164] J. Lee, P. Sharma, J. Tourrilhes, R. McGeer, J. Brassil, and A. Bavier, "Network integrated transparent tcp accelerator," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, April 2010, pp. 285–292.

[165] J. Cano, D. R. White, A. Bordallo, C. McCreesh, A. L. Michala, J. Singer, and V. Nagarajan, "Solving the task variant allocation problem in distributed robotics," *Autonomous Robots*, vol. 42, no. 7, pp. 1477–1495, Oct 2018. [Online]. Available: https://doi.org/10.1007/s10514-018-9742-5

[166] A. Andreyev, "Introducing data center fabric, the next-generation facebook data center network," *Facebook, Nov*, vol. 14, p. 13, 2014.

[167] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *ACM SIGCOMM computer communication review*, vol. 45, no. 4, pp. 183–197, 2015.