VNF-AAPC: Accelerator-aware VNF Placement and Chaining

Gourav Prateek Sharma, Wouter Tavernier, Didier Colle, Mario Pickavet

Email: gouravprateek.sharma@ugent.be
IDLab, Department of Information Technology
Ghent University - IMEC
Technologiepark-Zwijnaarde 126, 9052 Gent

1 Abstract

In recent years, telecom operators have been migrating towards network architectures based on Network Function Virtualization in order to reduce their high Capital Expenditure (CAPEX) and Operational Expenditure (OPEX). However, virtualization of some network functions is accompanied by a significant degradation of Virtual Network Function (VNF) performance in terms of their throughput or energy consumption. To address these challenges, use of hardware-accelerators, e.g. FPGAs, GPUs, to offload CPU-intensive operations from performance-critical VNFs has been proposed.
Allocation of NFV infrastructure (NFVi) resources for VNF placement and chaining (VNF-PC) has been a major area of research recently. A variety of resources allocation models have been proposed to achieve various operator's objectives i.e. minimizing CAPEX, OPEX, latency, etc. However, the VNF-PC resource allocation problem for the case when NFVi incorporates hardware-accelerators remains unaddressed. Ignoring hardware-accelerators in NFVi while performing resource allocation for VNF-chains can nullify the advantages resulting from the use of hardware-accelerators. Therefore, accurate models and techniques for the accelerator-aware VNF-PC (VNF-AAPC) are needed in order to achieve the overall efficient utilization of all NFVi resources including hardware-accelerators.
This paper investigates the problem of VNF-AAPC, i.e., how to allocate usual NFVi resources along-with hardware-accelerators to VNF-chains in a cost-efficient manner. Particularly, we propose two methods to tackle the VNF-AAPC problem. The first approach is based on Integer Linear Programming (ILP) which jointly optimizes VNF placement, chaining and accelerator allocation while concurring to all NFVi constraints. The second approach is a heuristic-based method that addresses the scalability issue of the ILP approach. The heuristic addresses the VNF-AAPC problem by following a two-step algorithm.
The experimental evaluations indicate that incorporating accelerator-awareness in VNF-PC strategies can help operators to achieve additional cost-savings from the efficient allocation of hardware-accelerator resources.

2 Keywords

Hardware-Accelerators; NFV; VNF; Placement; Chaining; Allocation; FPGA; GPU

3 Introduction

The incessant expansion in the number of connected users and network-services has resulted in exponential growth of traffic on the networks of telecom-operators. Telecom infrastructure thus needs to be scaled periodically to cope with the increasing traffic demands which result in high Capital Expenditure (CAPEX) and Operational Expenditure (OPEX). However, the growth in Average Revenue Per User (ARPU) has been very marginal due to the cut-throat competition among the operators. As a result, operators are forced to seek new network architectures that are scalable, agile and cost-efficient [1].
Network Function Virtualization (NFV) is a technology which leverages IT virtualization techniques for consolidating network appliances onto commercial-off-the-shelf (COTS) server machines. NFV aims to replace Network Functions (NFs) based on proprietary ASICs, also known as *middleboxes*, by their software instances running on

the general-purpose platforms consisting of x86 or ARM based high-volume servers (HVS). The software implementation of a NF running in a virtualized environment is called Virtual Network Function (VNF). Fig 1 shows the reference architecture of NFV as proposed by ETSI [2]. The purpose of the virtualization layer is to abstract the NFV Infrastructure (NFVi), which includes the compute, storage and networking resources, from VNFs running over it. Various virtualization technologies e.g. VMs, containers, are exploited for the realization of the virtualization layer.

Replacing network services based on middleboxes with VNF-chains running on COTS servers has several ad-

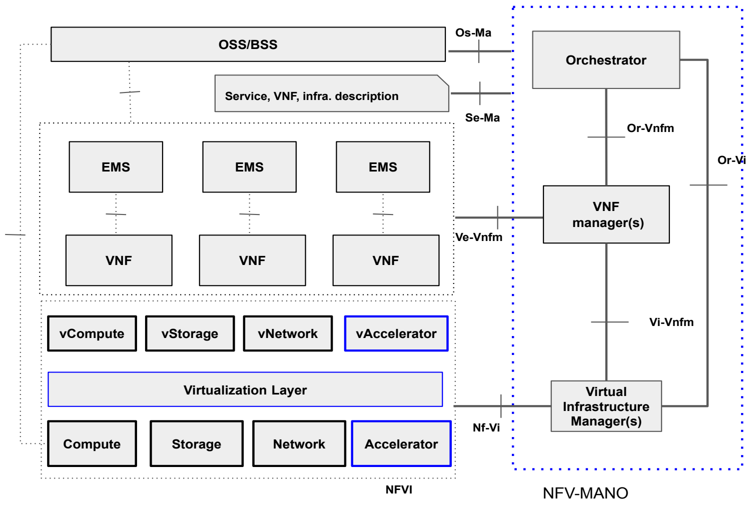

Figure 1: Reference NFV architecture by ETSI [2]. Components shown in blue needs to be added/updated as a result of inclusion of hardware-acceleration in NFVi.

vantages, such as the reduction in CAPEX and OPEX, faster time-to-market (TTM) of services, ease of service management and upgrade, etc [1]. Although, NFV offers several advantages, replacing NFs middleboxes with VNFs can have a detrimental effect on their packet-processing performance, e.g. loss of throughput and/or undeterministic latency. Furthermore, the growth in the computational capacity of CPUs is flattening with the time due to an expected end of Dennard's scaling and Moore's law in the coming years [3]. Performance improvement of software-based packet-processing platforms is expected to fall short as compared to the increasing data traffic on telecom networks. Therefore, matching the performance of middleboxes will be one of the key challenges faced by operators in the future too with regards to the widespread NFV adoption. This challenge has led to a recent interest in hardware-acceleration techniques for VNFs using externally connected hardware devices e.g. Graphics Processing Units (GPUs), Field Programmable Gate Arrays (FPGAs), Network Processing Units (NPUs), etc. Hardware-accelerators and CPUs can be used in conjunction such that CPU-intensive tasks can be offloaded from VNFs to hardware-accelerators and the rest of the VNF operations can be performed by the CPU of general-purpose hardware (COTS servers). As a consequence, an improvement in the overall packet-processing performance can be achieved.

Due to the upward trend of outsourcing network processing to the cloud, data centers (DCs) are being considered as NFVi. The share of energy costs in a DC, which includes the cost of energy spent in servers, switches and cooling of DCs, mainly constitute the OPEX cost. A large number of VNF CPU cycles are consumed in packet-processing tasks which otherwise consume a fraction of energy if implemented in the hardware. For example, using hardware-acceleration to offload iFFT/FFT in cloud-RAN (C-RAN) scenarios to FPGAs, GPUs or DSP can result in power saving by about 70% per carrier [4]. As a consequence, additional VNFs can be accommodated on the same NFVi as some CPU cores are freed because of the offload to hardware-accelerators.

Accelerators resources are being increasingly integrated with the NFVi layer along with the usual compute, network and storage (Fig. 1). However, the current Management and Orchestration (MANO) layer is mostly unaware of the acceleration requirements of VNFs and the location of hardware-accelerators in NFVi. As a result, the resource allocation decisions taken by the orchestrator are agnostic to VNF requirements and locality of hardware-accelerator resources. This could lead to sub-optimal utilization of NFVi resources. Particularly, the inefficient allocation of hardware-accelerator resources can negate the advantages resulting from the use of hardware-accelerators in NFV environments.

The overview of the accelerator-agnostic and accelerator-aware resource allocation procedure for VNF instantiation is depicted in Fig. 2 (a) and (b) respectively [2]. For the regular accelerator-agnostic VNF orchestration procedure (Fig. 2 (a)), NFV Orchestrator (NFVO) first validates the received VNF instantiation request and passes the corresponding VNF descriptor (VNFD) to the VNF Manager (VNFM). As VNFM is agnostic to accelerator requirement of the VNF or existence of any offload capability in NFVi, it requests the reservation of regular NFVi resources (compute, storage, and network) via Virtual Infrastructure Manager (VIM) which in turn allocates VMs/containers for the VNF and attach them to the network. VIM acknowledges NFVO when the resource reservation is complete. Further, deployment-specific configuration of VMs/containers can be performed through the corresponding VNFM after the VNF instantiation is completed. The instantiated VNF cannot offload its operations to a hardware-accelerator as it is not allocated any such special resource. However, NFVO can ask VIM to reserve hardware-accelerator resources for the VNF if it is aware of specific VNF requirements and the presence of offload capabilities in NFVi as shown in Fig. 2 (b). After processing the accelerator requirement mentioned in the VNFD, the VNFM requests resource allocation including hardware-accelerator resources. The instantiated VNF can now offload specific operations depending on the available types of accelerator implementations and amount of resources.

In order to achieve efficient utilization of all NFVi resource, it is imperative to incorporate accelerator-awareness

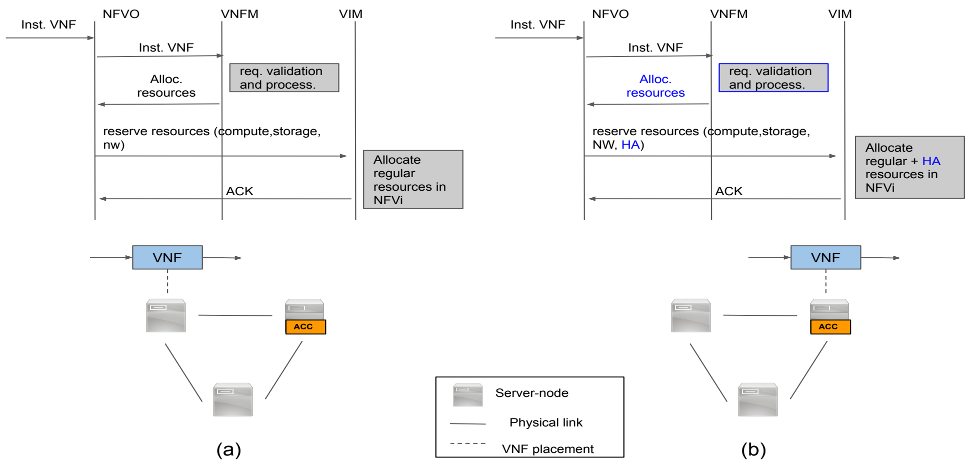
Inst. VNF
NFVO
VNFM
VIM
Inst. VNF
req. validation and process.
Alloc. resources
reserve resources (compute,storage, nw)
Allocate regular resources in NFVi
ACK
VNF
ACC
Inst. VNF
NFVO
VNFM
VIM
Inst. VNF
req. validation and process.
Alloc. resources
reserve resources (compute,storage, NW, HA)
Allocate regular + HA resources in NFVi
ACK
VNF
ACC
Server-node
Physical link
VNF placement
(a)
(b)

Figure 2: Processes involved in (a) accelerator-agnostic and (b) accelerator-aware VNF instantiation.

in the existing resource allocation models for NFV. VNF Placement and Chaining (VNF-PC) is the most important component of the NFV resource allocation procedure. VNF-PC is considered as NP-hard problem and has been widely topic researched in the literature [5]. With the inclusion of hardware-accelerator resources in NFVi, solving only the VNF-PC problem is not sufficient to obtain efficient allocation of NFVi resources. The VNF-PC problem needs to be altered in order to incorporate the resource allocation component for hardware-accelerators. We refer to this new problem as the Accelerator-aware VNF Placement and Chaining Problem (VNF-AAPC). Our objective in this paper is to model the VNF-AAPC problem and propose a scalable approach to solve this problem in a time-efficient manner. In order to address the above-mentioned objective we make the following contributions in this paper:

1. To obtain optimal solutions for the VNF-AAPC problem, we present an Integer Linear Program (ILP) formulation of this problem. It is a single-step exact method which jointly optimizes three decisions, namely– (i) VNF placement and (ii) chaining and (iii) accelerator allocation.

2. We design an efficient heuristic to solve the VNF-AAPC problem for DC topologies. This heuristic is particu-

larly useful for large-size instances of the VNF-AAPC problem where ILP model becomes too time-consuming to solve.

3. We also evaluate the ILP model and the proposed heuristic on two different data-center topologies. Furthermore, we compare the performance of accelerator-agnostic and accelerator-aware heuristics. Additionally, we present an analysis on the achievable cost-savings resulting from the use of hardware-accelerators in NFVi.

Section 4 of this paper deals with the discussion about hardware-acceleration in NFV environments. Relevant literature in the domain of NFV resource allocation is presented in Section 5. Section 6 describes the ILP formulation of the VNF-AAPC problem. The proposed heuristic to solve the VNF-AAPC problem is discussed in section 8. Performance evaluation results and comparison of the ILP model with our heuristic is reported in Section 9. Finally, future work and conclusions of this paper are presented in section 10.

4 Hardware-acceleration in NFV

The transition of telecom's network architectures from the purpose-built network appliances to VNFs running on COTS servers still face multiple challenges [1], [6]. One of the key obstacles is the virtualization of all NFs without breaking the Service Level Agreements (SLAs) of network services. However, it has been observed in many instances that the performance of VNFs is significantly degraded as compared to their hardware counterpart [7]. Authors in [7] investigated the impact of virtualizing firewall on its packet-processing performance. Processing latency in the virtual firewall could even reach ten times the processing latency in case of the hardware firewall. Performance bench-marking of IPSec is reported in a white paper by Intel in [8]. The results show that the processing of 48Gbps IPSec traffic requires on an average 9.5 CPU cores. The same traffic can, however, be processed using 4.6 CPU cores when accelerating AES-GCM de/encryption using a hardware-accelerator resulting in saving of about half of the CPU cores. Therefore, not just the performance boost of VNFs but also overall reduction in CPU utilization paves the way of hardware-accelerators in NFV environments.
A large number of VNFs involve CPU-intensive tasks like de-duplication, cryptography, compression, etc [8], [9], [10], [11], [12], [13], [14]. The software implementation of these tasks has been found to be very energy inefficient (numbers of operations performed/energy consumed) as compared to their hardware implementation resulting in excessive CPU utilization. The motivation behind using hardware-accelerators in NFV environments is that specific VNF components run more efficiently if implemented in hardware as opposed to a software running on a CPU of a general-purpose COTS servers. For example, GPUs have been used to speedup video-transcoding applications (H.264 and H.265) by 9.6x over software-only solution while being 6.4x more energy efficient [13].
Packet-processing in a VNF is usually accomplished by sequentially executing instructions of the VNF (VM/container) on one or more CPU cores. The packet-processing paradigm in architectures like FPGAs, NPUs, and GPUs is fundamentally different from that of a CPU. A GPU chip consists of thousands of computational cores that can be delegated execution units which are also known as GPU threads. Each GPU core executes the same NF on different packets sent by the CPU in the GPU memory [15]. With the large thread-level parallelism of GPUs and a good memory communication (low latency and high bandwidth), high packet-processing performance can be achieved. GPUNFV is a GPU-based NFV system which demonstrated line-rate packet-processing for stateful VNFs (e.g. flow monitor, firewall) by exploiting parallelism of GPUs [16]. FPGAs, on the other hand, contain millions of logic elements each of which contains lookup tables (LUTs) for implementing combinational logic and registers to store intermediary results. FPGAs also contain Block RAM (BRAM) to store a large amount of data which needs to be read (written) from (to) the main memory (RAM). Logic elements on an FPGA can be configured to realize different packet-processing functionalities. The parallelism in CPUs and GPUs is limited to the number of cores it has. Due to the massive amount of parallelism available on an FPGA at the gate-level, many processing tasks can be easily pipe-lined [17]. As a result, packet-processing tasks in VNFs can be offloaded to an FPGA very efficiently.
Hardware-acceleration can be applied to a variety of VNFs that can benefit from the different kinds of parallelism available on hardware-accelerators. Table 1 lists various VNFs alongside their sub-tasks that can benefit from offloading to hardware-accelerators. VNFs which contain components like cryptography, compression, de/encoding, etc, can be very efficiently offloaded to hardware-accelerators. Using an example of IPSec VNF, we next describe the most common approach of hardware-acceleration in NFV, i.e., FPGA look-aside acceleration.

Table 1: List of VNFs whose performance was improved after the indicated tasks were offloaded using hardware-accelerators.

VNF	Component functions for acceleration	References	Improvements
IPSec, SSH	AES en/decryption, SHA hash	[8], [9], [10], [11]	CPU usage reduction of 50% and 94% at packet size of 578B and 9000B, respectively [8].
DPI	Multihash, Bloomfilter, regex,	[11], [12]	20x throughput improvement [12].
Media Transcoding	VP8, H.264, H.256	[13]	9.6x gain in performance (FPS) and 6.4x overall efficiency (performance/watt).
vRAN	RS, FFT/iFFT, Turbo de/coding	[4], [14]	C-RAN power consumption reduction from 70W/carrier to 18W/carrier when i/FFT are offloaded. Turbo decoding time can be reduced by 50-60% by offloading it to a accelerator.
Dedup	Rabin hash, marker selection, chunk hash	[12]	8.2x improvement in throughput over software-only Dedup VNF.

4.1 VNF hardware-acceleration example

IPSec tunneling is one of the most popular ways of securing inter-network communication between branch-offices of an enterprise or LTE networks via encrypted tunnels [18]. Fig 3 (a) shows a standard IPSec setup. At one end of the IPSec tunnel, a VM containing IPSec application (e.g. libreswan [1]) is running on a server. The IPSec VNF must perform all the required cryptographic functions (en/decryption and SHA) on IPSec packets. These functions are usually provided by a software library (e.g. SSL) which contains implementations for various ciphers (e.g. DES-128, AES-128,256) and hashes (e.g. md5, SHA-256,512). Nowadays, certain CPU architectures (e.g. x86 and AMD) offer AES-NI and SHA-NI instructions dedicated for de/encryption and hashing operations which results in a better performance as compared to the traditional CPU architectures. Despite this improvement, a large number of CPU cores are still required to process the IPSec traffic at the line-rate, e.g., 9.5 CPU cores are required to handle IPSec traffic @ 48Gbps [8] as compared to only 3.3 CPU cores for processing of plain IP traffic (without IPSec). Moreover, packet-processing cost (CPU cycles/packet) varies with the packet-size which makes software-based IPSec solution inefficient for the IPSec packets of longer sizes ($>$ 1200 B).
Next, we describe the look-aside VNF hardware-acceleration approach taking IPSec as an example. A hardware designer typically first writes the required hardware-accelerator (e.g. AES-256, SHA-512) in a Hardware Descriptor Language (HDL), e.g. VHDL or Verilog. The HDL design is then compiled to a programming file, called bitfile using FPGA synthesis and implementation tools. The bitfile is then used to program the FPGA fabric in order to instantiate the desired accelerator function. The accelerator can then be modified or a new accelerator could be instantiated by re-programming the FPGA fabric with a bitfile corresponding to the new accelerator. This makes FPGAs re-programmable, unlike ASICs which offer a limited amount of configuration. In Fig. 3 (b), the AES (encryption and decryption) and SHA hash accelerators are instantiated by downloading their bitfiles to the FPGA card. Now, AES-256 de/encryption and SHA-512 hash operations can be offloaded from the IPSec VNF to accelerators running on the FPGA card [8]. For each IPSec packet, its payload is sent to the accelerator memory in order to perform the required cryptographic functions. After the function computation is over, the result of the operation is copied back from the accelerator memory to the main memory. The communication between the main memory and accelerators is accomplished via the PCIe bus. The overhead due to communications between CPUs and accelerators becomes insignificant for large packet-sizes. Moreover, hybrid chips like Intel Xeon+FPGA integrated-FPGA CPUs provide a tight coupling between CPUs and FPGAs thereby both CPUs and FPGAs can access the same memory and can avoid excessive overhead due to data transfers between them [19]. Nevertheless, many CPU cores are relieved from performing intensive cryptographic operations, thus a large number of CPU cores are free to run other workloads or VNFs [8].

[1]https://libreswan.org/

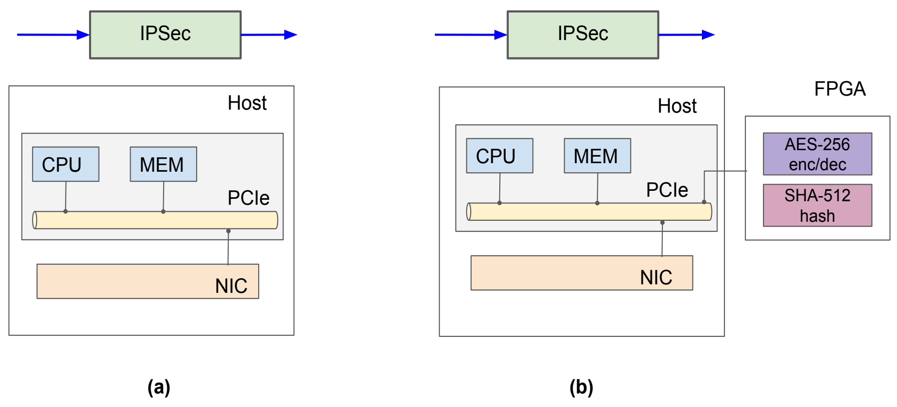

Figure 3: Illustration for the setup in (a) non-accelerated and (b) accelerated operation of IPSec VNF.

4.2 Trade-offs

Although highest programmability and flexibility can be achieved by running VNFs on CPUs (x86 or ARM), NF implementations based on technologies like GPUs, FPGAs, NPUs could be necessary for some performance-critical VNFs. Therefore, a spectrum of VNF implementation technologies result in a variety of solutions, ranging from one end of the highly-flexible and full-software NFs to the other end of the high-performance ASIC implementation and hardware-accelerated VNFs situated in between. Authors in [18] proposed an architecture for the unified handling and abstraction of hardware-accelerators in order to ease the manageability of accelerators. A virtual accelerator layer along with standard interfaces can be used in order to avoid compatibility and portability issues. This also helps to separate the concerns of VNF developers and hardware-accelerators designers. By abstracting hardware-accelerators, the same VNF image can be used for many hardware-accelerators without any modification. Fig. 4 illustrates the comparison between various VNF implementation technologies based on their performance and flexibility metrics [18]. A purpose-built ASIC implementation of an NF will offer the highest packet-processing performance but a very limited configuration will be possible e.g. update of forwarding tables in a router. On the other hand, platforms based on COTS servers offer huge programmability/flexibility, e.g. update of protocols, at the cost of performance. Although, devices having intermediate performance and flexibility, e.g. GPUs and FPGAs can also be used to realize full VNFs, however; more complex the packet processing task is, more challenging it is to implement on an FPGA or GPU. Hybrid platforms with a combination of CPU + hardware-accelerator (CPU+FPGA or CPU+GPU) are the most popular approach to achieving high-performance without losing too much programmability/flexibility. In hybrid platforms, the performance-critical tasks, e.g. en/decryption and hashing, etc, are implemented in the hardware and other complex tasks are still run in software running on a CPU. Keeping into account service requirements and trade-offs of various technologies, telecom operators or third-party VNF developers have to select the right platform for their VNF implementation. For example, IPSec VNF running on a CPU when offloaded to an FPGA can improve its throughput and halve its CPU usage with a fraction of more investment. ~~Due to their widespread popularity in the packet-processing application, we focus only on FPGAs as hardware-accelerators for VNFs. However, models and heuristics proposed in this paper could be easily adapted for different types of hardware-accelerators depending upon their nature.~~ There are two popular modes of using hardware-accelerators in the NFV environments, namely– look-aside and bump-in-the-wire [20]. Look-aside mode of hardware-acceleration is generally used to offload compute-intensive algorithms, e.g., offloading crypto-operations of IPSec to an FPGA. "Bump-in-the-wire" (in-line) is other mode where packet processing is done on the fly, e.g. on P4 switches or smartNICs, as they are transferred to/from the network. Bump-in-the-wire mode is therefore preferred mode to accelerate first/last VNFs of a VNF-chain [3] as accelerating VNFs. In this paper, our focus will be on modelling scenarios with look-aside mode of acceleration. However, to accommodate scenarios with bump-in-the-wire acceleration, appropriate constraints regarding required position-aware acceleration and latency requirements can be added to the proposed model.

Multiple VNFs running on a server-node can also share the same accelerator instance deployed on hardware-

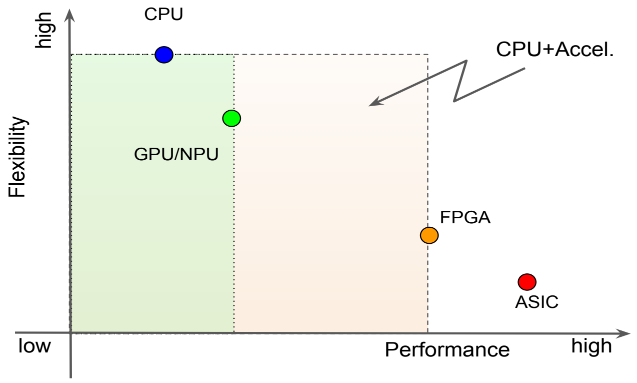

Figure 4: Comparison of various technologies for VNF implementation [18]. Green region: CPU+GPU and Orange region: CPU+FPGA.

accelerator cards. The network packets reaching a VNF running on a host are transferred to a particular accelerator instance over the PCIe bus. Offloading packet processing from stateless VNFs to accelerator instances is straightforward, as the order of incoming packet is not important. To offload stateful VNFs, where the state of a VNF is required to process packets, input packets along with the VNF state are transferred to the accelerator instance [16]. The state in the VNF is updated after the completion of processing in the accelerator instance.

5 Related Works

Various mathematical models and algorithms have been proposed to tackle the VNF-PC problem. The solution to VNF-PC problem attempts to allocate NFVi resources for the placement and chaining of VNFs. This problem is similar to Virtual Network Embedding (VNE) problem, a well-known problem in the area of network virtualization [5]. In VNF-PC problem, VNFs are equivalent to virtual nodes of VNE which are chained by virtual links. In addition to that, VNF-PC has an accompanying optimization goal which is described by the objective function of the problem. The objective function could be the minimization of power consumption, the required number of server-nodes, and links or maximization of resiliency, QoS, net profit, etc.

VNF-PC problem has been tackled using two different approaches in the past. The first approach is to exploit exact methods that result in an optimal solution but this approach is generally useful for small-scale instances of VNF-PC problem. Another approach ~~is~~ to solve VNF-PC problem is to use heuristics and thereby compromise a small amount of efficiency for the scalability.

Using the ILP formulation, authors in [21] modeled resource allocation in a hybrid NFV scenario where services are provided using both dedicated hardware appliances and VNFs. This model was evaluated using two types of service chain requests and a small service provider scenario. A Mixed Integer Quadratically Constrained Program (MIQCP) model for VNF-PC optimization problem was introduced in [22]. Pareto set analysis was performed to investigate the trade-offs between three different objective functions. The evaluation of the model shows the objective function (e.g. minimization of latency, link utilization or allocated nodes) has a direct impact on the VNF placement and chaining. Authors in [23] formulated the multi-objective VNF-PC problem considering both legacy Traffic Engineering (TE) ISP goals and combined TE-NFV goals.

Because of the inherent complexity of VNF-PC problem, exact approaches based on ILP/MILP become impractical for realistic network sizes. Therefore, many heuristic-based algorithms have also been proposed to solve this problem in a reasonable time.

The problem of Elastic VNF Placement(EVNFP) was studied in [24] and an ILP model was presented for minimizing operational costs in NFV scenarios. Authors also developed an algorithm called Simple Lazy Facility Location (SLFL) in order to solve EVNFP problem in polynomial time. Evaluations show that SLFL reduced operational costs by 5-8% and also increased the request acceptance rate by 2x as compared to the first-fit and random alternatives.

S. Sahhaf et al. studied the decomposition and embedding of network services in [25]. An ILP model was proposed whose objective was to minimize the total cost due to the mapping of different decomposed VNF components (e.g. VM, container, DPDK) to the physical nodes in NFVi. A heuristic algorithm, consisting of two phases– backtracking and mapping, was also proposed. The experimental results show a decrease in mapping cost and an increase in the request acceptance ratio in the long run for both ILP and heuristic approaches.

F. Carpio et al. studied the problem of network load balancing for the deployment of Service Function Chains (SFCs) [26]. In particular, the authors addressed the problem of distance-to-data center by the use of VNF replicas in order to load balance the network. Three approaches– ILP model, Genetic Algorithm, and random fit placement algorithm were designed and compared to realize efficient VNF placement and replication method in an NFV environment.

Although a lot of resource allocation studies have been carried out in the past, only two studies have considered hardware-accelerator in their models. H. Fan et. al. proposed an architecture to implement uniform deployment and allocation of accelerator resources in NFV environments [27]. The authors proposed an algorithm to achieve efficient allotment of accelerator resources in forwarding and server nodes. Algorithms take as an input the network topology and capacity of physical resources and output the amount of accelerator resources that should be provided on forwarding or server nodes. This study concerns the optimization of accelerators resource provisioning not with the optimization of accelerator allocation to VNFs.

The concept of heterogeneous components has been described in [28]. A heterogeneous service consists of multiple implementation options that could be deployed to serve the dynamic requirements of the service. The paper studied the problem of joint Scaling, Placement and Routing (SPRING) for heterogeneous services. To address the SPRING problem, a MILP formulation and a heuristic algorithm were proposed. The SPRING model focuses on efficient resource allocation in heterogeneous infrastructure with lower processing times. This paper does not consider the distribution of hardware-accelerator resources in a data-center. Furthermore, VNF-PC decision did not take into account the communication between the hardware-accelerator and the CPU on a server-node.

This work is a major extension to our previous work where we modeled only VNF placement in a heterogeneous NFV environment [29] using a best-fit based approach. Here, we address the complete problem of accelerator-aware VNF placement and chaining along with a thorough evaluation of the ILP model and heuristics.

6 Problem Overview

Services in the NFV domain are realized by processing network traffic through a sequence of VNFs. In order to fully exploit the benefits of NFV technology, it is necessary to efficiently allocate NFVi resources to VNF-chains. Resource allocation requires a mapping of the service's VNF Forwarding Graph (VNF-FG) to NFVi resources [5]. A VNF-FG consists of nodes representing VNFs and edges stand for virtual links between VNFs. Therefore, the mapping process can be thought of as a two steps process, namely (i) VNF placement and (ii) VNF Chaining. "VNF placement" involves the assignment of VNFs to COTS servers, whereas "VNF chaining" step involves allocation of a path in the physical network to every virtual link of VNF-FG. "VNF chaining" ensures the appropriate steering of network traffic through the sequence of VNFs constituting the service. Together this problem is referred to as VNF placement and chaining (VNF-PC) problem.

In addition to the usual compute, network and storage resources, NFVi also includes hardware-accelerator resources. With the inclusion of hardware-accelerators in NFVi, VNF-PC models must be revised. In order to ensure efficient utilization of all NFVi resources, both placement and chaining decision should take into account the accelerator resources (e.g. total logic elements and BRAM of FPGAs, cores/threads of GPUs) along with the usual NFVi resources, i.e. compute, storage and network. This problem will be referred to as accelerator-aware VNF placement and chaining (VNF-AAPC) problem.

We motivate the importance of modeling the VNF-AAPC problem by a simple example illustrated in Fig 5. As an input, NFVi consists of five server-node each with 5 CPU cores and connected with each other as shown in Fig 5. One of the server-node is equipped with a hardware-accelerator card connected over the PCIe bus. The objective of VNF-PC problem is to deploy VNF-chains s_1 and s_2 using as few server-nodes as possible. The CPU requirements of all VNFs is indicated in the boxes above each VNF. VNF f_{12} is an 'accelerate-able' VNF, i.e., it consumes 4 CPU units when it is not accelerated and 2 CPU units when it is able to offload its operations to an accelerator on a hardware-accelerator card. For the sake of simplicity, we assume sufficient bandwidth is available on physical links for the chaining of VNFs. The result of the usual (accelerator-agnostic) VNF placement method, where only CPU resources are considered, is shown in Fig. 5 (a). In total, five server-nodes are required for the deployment of s_1 and s_2. With the accelerator-aware strategy, however, only four server nodes are required for the placement of same VNF-chains as shown in Fig 5 (b). This is because the VNF f_{12} is deployed on a server-node attached with a hardware-accelerator card and is able to reduce its CPU requirement by half.

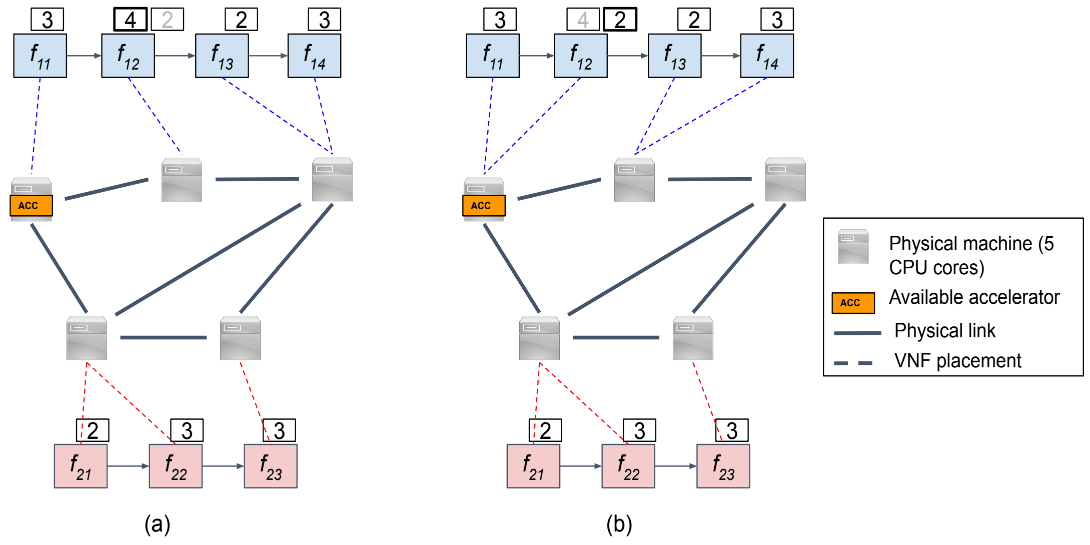

Figure 5: Illustration comparing VNF placement in accelerator-agnostic and accelerator-aware VNF placement scenarios. The CPU requirement of each VNF is indicated in the box above it.

7 ILP Formulation

Next, we introduce notations, decision variables, objective function and constraints required for the ILP formulation of the VNF-AAPC problem. ILP model for the VNF-AAPC problem provides a single-step method for obtaining optimal resource allocation.
Table 2 gives the description of the notations used in the formulation. NFVi network is represented by a connected directed graph $G = (N, E)$. Set N consists of all the physical nodes in the NFVi network and E represents all the physical links between nodes. A node can be a computational device (e.g. COTS server) or a forwarding device (e.g. switch). $N^c \subset N$ denotes a set of all COTS servers having computational resources required to run VNFs.
The capacity of different resource types in a node $n \in N^c$ is denoted by the following three parameters: $\mathcal{R}_{cpu}(n)$, $\mathcal{R}_{bus}(n)$ and $\mathcal{R}_{acc}(n)$. $\mathcal{R}_{cpu}(n)$ denotes the total number of CPU cores available for running VNFs on node n.
IO communication capacity of a node is dependent on the bandwidth (Mbps) of the PCI(e) bus which is denoted by $\mathcal{R}_{bus}(n)$. The same PCI(e) bus is shared for two tasks. First, for communication with accelerators and secondly for sending/receiving packets to/from the network using NIC card.
The amount of resources present on the hardware-accelerator card attached to server-node n is represented by $\mathcal{R}_{acc}(n)$. We use $\mathcal{R}_{acc}(n)$ only to denote the total number of logic elements present on an FPGA board. However, other resources like the amount of BRAM available on an FPGA can also be represented similarly.
A is a catalog of all types of accelerator implementations available for instantiation on the hardware-accelerator cards attached to server-nodes. For example, if FPGA bitfile implementations for only AES and SHA accelerators are available, i.e. $A = \{AES, \ SHA\}$, en/decryption (AES) or hashing (SHA) tasks from an IPSec VNF can be offloaded to AES and SHA accelerators running on an FPGA card. However, tasks like en/decoding involved in the vTC (video transcoding) VNF cannot be offloaded using any accelerator implementation present in A.
Each implementation of an accelerator type $a \in A$ requires a certain amount of resources, represented by $r(a)$, on the hardware-accelerator card. Again, $r(a)$ can be used to represent requirement of any type of resource on a hardware-accelerator card. In our formulation, $r(a)$ denotes only the required number of logic elements to implement an accelerator of type a on an FPGA card.
We assume each service-request s received by the telecom operator consists of a VNF-FG $\mathcal{G}^s$ and corresponding bandwidth requirement (t_s). The VNF-FG $\mathcal{G}^s = (\mathcal{F}^s, \mathcal{L}^s)$ of the service-request s consists of a set of VNFs $\mathcal{F}^s$ and

a set of virtual links between VNFs denoted by $\mathcal{L}^s$. Two consecutive VNFs of the service-request s denoted by f^s_k and f^s_{k+1} are joined by a virtual link $(f^s_k, f^s_{k+1}) \in \mathcal{L}^s$. For the sake of simplicity, we assume the traffic compression ratio of every VNF is 1. This implies that the amount of traffic (t_s) doesn't change while passing through a sequence of VNFs.
CPU requirement for a VNF $f^s \in \mathcal{F}^s$, in terms of the total number of cores required, is denoted by $cpu_0(f^s)$ and the reduction in the total number of cores due to offloading is denoted by $cpu_r(f^s)$. In other words, $cpu_0(f^s)$ denotes the number of CPU cores required by the VNF f^s to process the network traffic coming at the rate t_s.
The type of accelerator required for offloading a VNF f^s is denoted by $atype(f^s)$.
$\alpha^n_{f^s}$ is a binary variable used to indicate if VNF f of the service-request s is placed on node n. Allocation of an accelerator to VNF f^s placed on node n is indicated by $\beta^n_{f^s}$. A computational node $n \in N^c$ is said to be in-use if at least one VNF is placed on n. This is denoted by a binary variable x_n. Instantiation of an accelerator of type a on the hardware-accelerator card attached to n is indicated by a binary variable δ^n_a.
The binary variable $\gamma^{n_i,n_j}_{f^s_k,f^s_{k+1}}$ is an indicator variable which denotes if the virtual link $(f^s_k, f^s_{k+1}) \in \mathcal{L}^s$ mapping to a path in G contains physical-link (n_i, n_j) or not.
In a scenario when a telecom operator leases server-nodes from an Infrastructure Provider (InP) to deploy VNF-chains, she ought to acquire a minimum number of server-nodes as possible. The cost of a server-node is included to the total cost if that node is used to host at least one VNF. The cost of using a computational node $n \in N^c$ in is denoted by C_n (in \$). Usually, parameters like $\mathcal{R}_{cpu}(n)$, $\mathcal{R}_{bus}(n)$ and $\mathcal{R}_{acc}(n)$ determine the value of c_n.

Table 2: Description of parameters and decision variables

Input parameters	
Notation	Description
G	Directed graph $G = (N, E)$ represents the network.
N	Set of all forwarding and computational nodes within the network.
N^c	Set $N^c \subset N$ contains all nodes of the network with positive computational resources (all server-nodes).
$b(n_i, n_j)$	Maximum bandwidth (in Mbps) of a physical-link $(n_i, n_j) \in E$. **
$\mathcal{R}_{cpu}(n)$	Maximum CPU resources (in total number of CPU cores) available on $n \in N^c$.
$\mathcal{R}_{acc}(n)$	Maximum accelerator-fabric resources (in total number of logic elements) available on $n \in N^c$.
$\mathcal{R}_{bus}(n)$	Maximum bandwidth (in Mbps) of the PCIe bus of node $n \in N^c$.
A	Set of all available accelerator types (in NFVI).
$r(a)$	Resource requirement (logic elements) of the accelerator type $a \in A$.
S	Set of all VNF-chains.
$\mathcal{G}^s$	Directed graph $\mathcal{G}^s = (\mathcal{F}^s, \mathcal{L}^s)$ represents VNF-FG of request $s \in S$.
$\mathcal{F}^s$	Set of all VNFs in VNF-FG of the VNF-chain $s \in S$.
$\mathcal{L}^s$	Set of all directed virtual links in the VNF-FG of the VNF-chain $s \in S$.
t_s	Throughput requirement (Mbps) of the VNF-chain $s \in S$.
$cpu_0(f^s)$	CPU requirement (cores) of VNF $f \in \mathcal{F}^s$.
$cpu_r(f^s)$	CPU reduction (cores) for VNF $f \in \mathcal{F}^s$.
$atype(f^s)$	Type of accelerator needed for acceleration of VNF $f\mathcal{F}^s$.
c_n	Cost (\$) of running a computational node $n \in N^c$.

Decision variables	
Notation	Description
$\alpha^n_{f^s}$	Binary variable indicates if VNF f^s of VNF-chain s is placed on n.
$\beta^n_{f^s}$	Binary variable indicates if VNF f^s of VNF-chain s is accelerated on n.
x_n	Binary variable indicates if computational node $n \in N^c$ is used for hosting at-least one VNF.
δ^n_a	Binary variable indicates if accelerator of type a is instantiated on the node n
$\gamma^{n_i,n_j}_{f^s_k,f^s_{k+1}}$	Binary variable indicates if the virtual link (f^s_k, f^s_{k+1}) mapping to a path in the physical-network contains the physical-link (n_i, n_j), $(n_i, n_j) \in E$.

Next, we discuss the objective function and constraints describing the ILP model for the accelerator-aware VNF placement and chaining problem.

7.1 Objective

The objective (1) of our ILP formulation is to minimize the total cost incurred to the operator from the use of server-nodes, some of which are attached to a hardware-accelerator card. The decision variable x_n is used to determine whether a server-node is used or not.

$$obj : \min \Big(\sum_{n \in N^c} c_n x_n \Big) \tag{1}$$

7.2 Constraints

We classify all the constraints in four categories: (i) Physical node constraints, (ii) Link Mapping constraints, (iii) Accelerator Constraints and (iv) Auxiliary Constraints, which are explained as follows.

7.2.1 Physical Node Constraints

The sum of effective CPU usage of all VNFs placed on any node should not surpass its maximum CPU capacity. This constraint in depicted in (2).
The constraint in (3) indicates the finite availability of resources on the hardware-accelerator card for the instantiation of accelerators.
The rate of communication between VNFs and accelerators instantiated on the hardware-accelerator card is bounded by the maximum bandwidth of the PCIe bus, as indicated in (4). The first term in the LHS of (4) is the bus bandwidth consumption due to the traffic between neighboring VNFs. First, summation over the traffic coming from VNFs (f_k^s) placed on server-node n_i to its neighboring VNFs (f_{k+1}^s) placed on n_j is carried out and a factor of two is there to represent the traffic both coming to and from the VNFs running on server-node n_i. The term $2t_s\beta_{f^s}^{n_i}$ represents the bandwidth utilization due to communication between the VNF f^s and accelerator fabric on the node n.

$$\sum_{s \in S, f^s \in \mathcal{F}^s} \alpha_{f^s}^n cpu_0(f^s) - \beta_{f^s}^n cpu_r(f^s) \leq \mathcal{R}_{cpu}(n) \quad \forall n \in N^c \tag{2}$$

$$\sum_{a \in A} r(a)\delta_a^n \leq \mathcal{R}_{acc}(n) \quad \forall n \in N^c \tag{3}$$

$$\sum_{\substack{\forall n_j \in N \\ ((n_i,n_j) \in E)}} \sum_{\substack{s \in S, \\ (f_k^s, f_{k+1}^s) \in \mathcal{L}^s}} 2t_s \gamma_{f_k^s, f_{k+1}^s}^{n_i,n_j} + \sum_{\substack{s \in S, \\ f^s \in \mathcal{F}^s}} 2t_s \beta_{f^s}^{n_i} \leq \mathcal{R}_{bus}(n_i) \quad \forall n_i \in N^c \tag{4}$$

7.2.2 Physical link constraints

The flow-conservation constraint is described in (5). This constraint ensures that a virtual link (f_k^s, f_{k+1}^s) is always mapped to a physical path in the network. Also, it ensures that for a non-computation node $n \in N \setminus N^c$, the net-traffic outflow or inflow is always zero.
The constraint in (6) guarantees that the sum of bandwidths allocated to virtual links on a physical-link (n_i, n_j) never exceeds its capacity $b(n_i, n_j)$.

$$\begin{gathered} \sum_{\substack{\forall n_j \in N \\ ((n_i,n_j) \in E)}} (\gamma_{f_k^s, f_{k+1}^s}^{n_i,n_j} - \gamma_{f_k^s, f_{k+1}^s}^{n_j,n_i}) = (\alpha_{f_k^s}^{n_i} - \alpha_{f_{k+1}^s}^{n_i}) \\ \forall s \in S, \forall (f_k^s, f_{k+1}^s) \in \mathcal{L}^s, \forall n_i \in N \end{gathered} \tag{5}$$

$$\sum_{\substack{s \in S \\ (f_k^s, f_{k+1}^s) \in \mathcal{L}^s}} t_s \gamma_{f_k^s, f_{k+1}^s}^{n_i,n_j} \leq b(n_i, n_j) \quad \forall (n_i, n_j) \in E \tag{6}$$

7.2.3 Accelerator constraints

The constraint in (7) is a consequence of the fact that a VNF f^s can be given access to an accelerator on a node n only if it is placed on it.
The constraint in (8) ensures that an accelerator of a particular type is instantiated if a non-zero number of VNFs are using that accelerator type. This constraint is easily linearized by replacing it with a pair of constraints indicated in (9a-9b). M_1 (big M) in constraint (9b) is a constant with a value greater than the total number of VNFs f^s in all the service-chain requests $s \in S$.

$$\beta^n_{f^s} \leq \alpha^n_{f^s} \quad \forall n \in N, \forall s \in S, \forall f^s \in \mathcal{F}^s \tag{7}$$

$$\delta^n_a = \begin{cases} 1, & \text{if} \sum\limits_{\substack{\forall s \in S, \forall f^s \in \mathcal{F}^s, \\ a = atype(f^s)}} \beta^n_{f^s} \geq 1 \\ 0, & \text{otherwise} \end{cases} \qquad \forall n \in N, \forall a \in A \tag{8}$$

$$\delta^n_a \leq \sum_{\substack{\forall s \in S, \forall f^s \in \mathcal{F}^s \\ a = atype(f^s)}} \beta^n_{f^s} \quad \forall n \in N, \forall a \in A \tag{9a}$$

$$\sum_{\substack{\forall s \in S, \forall f^s \in \mathcal{F}^s \\ a = atype(f^s)}} \beta^n_{f^s} \leq M_1 \delta^n_a \quad \forall n \in N, \forall a \in A \tag{9b}$$

7.2.4 Auxiliary Constraints

The set of constraints in this subsection restrict the value of decision variables x_n, ~~y_{n_i,n_j}~~, $\alpha^n_{f^s}$, β_{f^s}, δ^n_a, $\gamma^{n_i,n_j}_{f^s_k,f^s_{k+1}}$.
A server-node is considered to be running if at least one VNF is mapped onto it, as indicated by the constraint in (10). The pair of constraints (11a - 11b) forces x_n to be equal to 1 if at least one VNF is placed on node n. In constraint 11b, M_2 is a constant with a value greater than the total number of VNFs f^s in all service-chain requests $s \in S$.

$$x_n = \begin{cases} 1, & \text{if} \sum\limits_{\forall s \in S, \forall f^s \in \mathcal{F}^s} \alpha^n_{f^s} \geq 1 \\ 0, & \text{otherwise} \end{cases} \qquad \forall n \in N \tag{10}$$

$$x_n \leq \sum_{s \in S, f^s \in \mathcal{F}^s} \alpha^n_{f^s} \quad \forall n \in N \tag{11a}$$

$$\sum_{s \in S, f^s \in \mathcal{F}^s} \alpha^n_{f^s} \leq M_2 x_n \quad \forall n \in N \tag{11b}$$

Each VNF in a service-chain request must be placed only once. This is represented by the constraint in (12). The set of constraint in (13) ensures decision variables $\alpha^n_{f^s}, \beta^n_{f^s}, \delta^n_a$ and $\gamma^{n_i,n_j}_{f^s_k,f^s_{k+1}}$ can only take binary (0 or 1) values.

$$\sum_{n \in N} \alpha^n_{f^s} = 1 \quad \forall s \in S, \forall f^s \in \mathcal{F}^s \tag{12}$$

$$\begin{gathered} x_n, \alpha^n_{f^s}, \beta^n_{f^s}, \delta^n_a, \gamma^{n_i,n_j}_{f^s_k,f^s_{k+1}} \in \{0,1\} \\ \forall n \in N, \forall (n_i,n_j) \in E, \forall s \in S, \forall f^s \in \mathcal{F}^s, \forall (f^s_k, f^s_{k+1}) \in \mathcal{L}^s \end{gathered} \tag{13}$$

The above ILP formulation implements a single-step method to solve the VNF-AAPC problem. For a given NFVi graph G and a set of requested service-chain requests S, the above ILP formulation not only given an optimum VNF placement $\alpha^n_{f^s}$ and chaining $\gamma^{n_i,n_j}_{f^s_k,f^s_{k+1}}$ solution but also gives an optimum accelerator allocation $\beta^n_{f^s}$ for VNFs. As the VNF-PC is considered to be an NP-hard problem, it does not scale with the problem size. The accelerator awareness further increases its complexity. As a result, the ILP formulation of the VNF-AAPC problem is challenging to solve for networks of realistic sizes. In order to address the non-scalability issue with ILP, we propose a method based on heuristics for solving the VNF-AAPC problem in a time-efficient manner.

8 Proposed Heuristics

Next, we describe two heuristic-based algorithms for solving the VNF-PC problem for NFVi containing hardware-accelerator resources along with the usual resources. The first heuristic we propose is an accelerator-agnostic algorithm which does not take into the account presence of hardware-accelerators in NFVi while performing VNF-FG mapping. This algorithm will serve as a baseline for the evaluation of our second algorithm, i.e., accelerator-aware VNF-PC heuristic.

8.1 Accelerator-agnostic VNF-PC heuristic

Accelerator-agnostic VNF-PC heuristic involves the hierarchical deployment of VNF-chains [30]. Hierarchical deployment exploits classification of nodes into different levels of DC topologies, e.g. different levels in a leaf-spine DC topology are server, rack, and cluster. Starting from the lowest level, i.e. server-node level, VNF-PC is attempted at each level until the VNF-chain is deployed at a level. Also, previously used server-node is checked for the placement of subsequent VNF of a VNF-chain resulting in the localization of VNFs of the same VNF-chain.
The pseudo-code for the accelerator-agnostic VNF-PC algorithm is described in Alg. 1. The procedure `AgPlaceChain` is called from Alg. 2 in order to map VNF-FG ($G^s = (\mathcal{F}^s, \mathcal{L}^s)$) corresponding to all service-requests (line 3) onto NFVi. The mapping of each VNF-FG is attempted at different levels of the data-center, e.g., in leaf-spine topology, first at *node*, then at *rack*, and at last at *cluster* level (Alg. 2 lines 2-6). Mapping on *node* level is done by assigning $NodesSet$ equal to the set of all server-nodes $\forall n \in N^c$. If no one server-node is able to allocate all the VNFs of a chain, $NodesSet$ is assigned all nodes per *rack*. If the mapping of a VNF-chain is not possible to any of the *rack*, $NodesSet$ is assigned all the nodes in the *cluster* and VNF-PC is attempted again.
In Alg. 1, for each VNF $f^s \in \mathcal{F}^s$ placement is first tried on the previously used node n_p. A new node is only selected if enough CPU resources aren't available on n_p (line 7-15). An attempt for accelerator allocation is done on node n_p (line 16) by invoking procedure `AccelVNF`. When all VNFs $\forall f^s \in \mathcal{F}^s$ are placed, virtual-links are mapped to physical-paths in G using the procedure `ChainVNFs`. If the placement of any VNF $f^s \in \mathcal{F}^s$ fails or procedure `ChainVNFs` returns **False**, all resources are updated to their previous values just after the start of the procedure `AgPlaceChain` (lines 23-25).
The procedure `AccelVNF` (Alg. 3) checks whether an accelerator can be granted to VNF f^s node n_p. This is done by verifying whether enough CPU and bus resources are available on n_p (line 2). If $atype(f^s)$ is not already instantiated on the hardware-accelerator card attached to node n_p, it is checked whether enough accelerator resources are available on the card (lines 5-11) to instantiate the accelerator type $atype(f^s)$. All the required resources are updated accordingly if f^s is allocated an accelerator (lines 9-10,13-14) in this procedure.
The chaining procedure for mapping of virtual links to physical paths is described in Alg. 4. For each virtual link $(f_k^s, f_{k+1}^s) \in \mathcal{L}^s$, all set of shortest paths between two physical nodes hosting f_k^s and f_{k+1}^s are stored first in P (line 7). Each path is checked sequentially for its available bandwidth on all of its physical links using the procedure `bw` (lines 9). If a path with enough bandwidth is available, $\gamma[f_k^s, f_{k+1}^s]$ (line 13) along with bus (line 12) and link bandwidths (line 11) are updated for every physical link ($\forall(n_i, n_j) \in p$) in the path p. If any virtual link cannot be mapped to a physical path, values of resources and variables are reverted to their previous values at the start of the procedure (lines 19-22).
An example illustrating the working of the accelerator-agnostic VNF-PC heuristic is shown in Fig. 6. Consider two VNF-chains ($s_1 : f_{11} \rightarrow f_{12} \rightarrow f_{13} \rightarrow f_{14}$, $s_2 : f_{21} \rightarrow f_{22} \rightarrow f_{23}$) supposed to be deployed on a given NFVi, which here is a DC in the leaf-spine topology. The heuristic starts with the chain s_1 and first tries its deployment on the server-node level. As no server-node can accommodate all VNFs of this VNF-chain, the heuristic moves to the next level of the topology, i.e., rack level. Again, no rack has enough resources to host the complete VNF-chain s_1. Therefore, the heuristic now considers all server-nodes of the cluster and uses $rack0$ and $rack1$ for the placement of VNF-chain s_1. After the placement of all VNFs of the first chain is completed, network bandwidth is then allocated to the virtual links of the VNF-chain via the `ChainVNFs` procedure as shown in Fig. 6. The same process will be followed for the deployment of the VNF-chain s_2 which is deployed in $rack2$.

8.2 Accelerator-aware VNF-PC heuristic

The accelerator-aware VNF-PC heuristic ~~is based on~~ combines hierarchical deployment with segmentation of VNF-chains. A VNF-chain is first split at 'accelerate-able' VNFs, i.e. VNFs which require hardware-accelerators $\forall s, \forall f^s, atype(f^s) \in A$. VNF-chain deployment is then performed in two phases. In the first phase, VNF placement along with accelerator-allocation is performed for all accelerate-able VNFs. Sub-graphs (VNF-FGs) corresponding

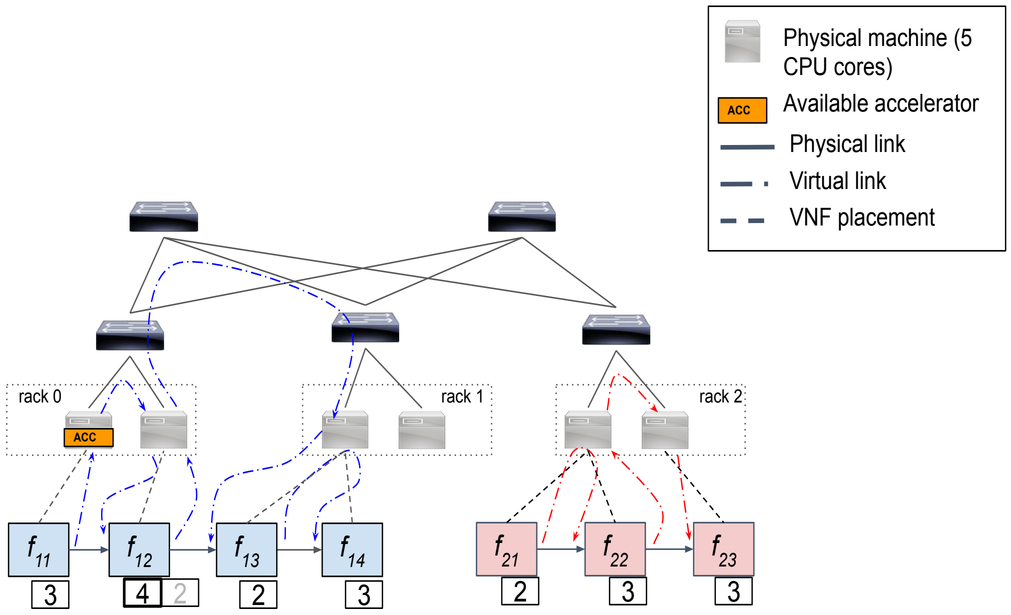

Figure 6: Illustration showing placement and chaining in accelerator-agnostic VNF-PC heuristic on the leaf-spine topology.

to the remaining VNF-chain segments are mapped to the NFVi using the hierarchical deployment in the second phase.
The procedure for allocation of accelerators to VNFs (`PlaceAccelVNFs`) is shown in Alg. 5. First, a list of all server-nodes attached with a hardware-accelerator card are stored in N_a. Accelerate-able VNFs constituting the VNF-chain request s are assigned to F^s_{acc}. Accelerator allocation is then attempted for every VNF in F^s_{acc}. A list of server-nodes with enough resources and having accelerator $atype(f^s)$ already instantiated on its attached hardware-accelerator card is stored in N_{f^s} (line 6). Out of all server-nodes in N_{f^s}, a node, a closest node where (any) previous VNF of the same VNF-chain request s was placed, is assigned to n_a (line 8-9). If no node has sufficient resources and accelerator of type $atype(f^s)$ instantiated on it, a node with highest CPU utilization is then selected from N_a (line 11). Using the procedure `AccelVNF` (Alg. 3), placement and accelerator allocation for VNF f^s is attempted on n_a (line 13).
In Alg. 6, each server-node used in previous step is iterated over for the complete mapping of remaining VNF-chain segments (lines 4-27). Un-mapped segments of all service-requests are identified for which at least one adjacent VNF is placed on node n (lines 5-9). An attempt is then made to map each segment seg in S^n_{seg} with as much proximity to n as possible. The process followed for the mapping of each VNF-FG segment $\forall seg \in S^n_{seg}$ is similar to the one followed in the accelerator-agnostic VNF-PC heuristic (Alg. 2). The mapping is attempted first on *nodel* level, then on the *rack* level containing *node* and at last on the whole *cluster* level using the procedure `AgPlaceChain` (line 14). In addition, newly placed VNF segment seg and its adjacent VNFs, which were previously placed using `PlaceAccelVNFs`, are linked via procedure `ChainVNFs` (line 19).
At last, VNF-chain requests which haven't been yet mapped to NFVi are identified (line 28). Set S_R contains all those VNF-chain requests $s \in S$ which either (i) do not have any VNF with an accelerator implementation available in A or (ii) enough resources were not available to allocate accelerator to VNFs during the first step (line 2). Mapping of all service-requests in S_R is attempted in the hierarchical way (lines 31-35) discussed in Alg. 2.
Again, consider the deployment of two VNF-chains ($s_1 : f_{11} \to f_{12} \to f_{13} \to f_{14}$, $s_2 : f_{21} \to f_{22} \to f_{23}$) on the same NFVi topology as shown in Fig. 7. In accelerator-aware PC heuristic, accelerate-able VNFs of two VNF-chains are placed in the first phase, so f_{12} is placed on the first server-node of $rack0$ which has an attached hardware-accelerator card. In the second phase, the heuristic loops over the server-nodes which have VNFs placed on them, while determining and placing the remaining segments of VNF-chains. Therefore, deployment of the VNF-chain segment f_{11} and $f_{13} \to f_{14}$ is then attempted using the same procedure as discussed in accelerator-agnostic heuristic. After the successful deployment of VNF-chain segments of s_1, two segments f_{11}, $f_{13} \to f_{14}$ are chained to the VNF f_{12} via `ChainVNFs` procedure. At last, the second VNF-chain (no accelerate-able VNFs) s_2 is then deployed

Algorithm 1: Accelerator-agnostic VNF-PC procedure.

```
1  Procedure AgPlaceChain($NodesSet,\ \alpha,\ \beta,\ \gamma,\ (\mathcal{F}^s, \mathcal{L}^s)$):
2      $tries, plc, n_p \leftarrow 0,$ True$, \phi$;
3      while $tries \leq MAX\_TRIES$ do
4          for $Nodes$ in $NodesSet$ do
5              $\alpha_0, \beta_0, Nodes_0 \leftarrow \alpha, \beta, Nodes$;
6              for $f^s$ in $\mathcal{F}^s$ do
7                  if $n_p == \phi$ or $\big(cpu_0(f^s)\big) > \mathcal{R}_{cpu}(n_p)\big)$ then
8                      $N = \{n : \forall n \in Nodes, \mathcal{R}_{cpu}(n) \geq (cpu_0(f^s)\}$;
9                      if $N == \phi$ then
10                         $plc \leftarrow$ False;
11                         break;
12                     else
13                         $n_p \leftarrow$ Random($N$)*;
14                     end
15                 end
16                 if AccelVNF($f^s, n_p,\ node\_accels,\ t_s$) == False then
17                     $\mathcal{R}_{cpu}(n_p) \leftarrow \mathcal{R}_{cpu}(n_p) - cpu_0(f^s)$;
18                 else
19                     $\beta[f^s] \leftarrow n_p$;
20                 end
21                 $\alpha[f^s] \leftarrow n_p$;
22             end
23             if ChainVNFs($\alpha,\ \gamma,\ \mathcal{L}^s,\ G$) == False or $plc$ == False then
24                 $\alpha, \beta, Nodes \leftarrow \alpha_0, \beta_0, Nodes_0$;
25             else
26                 return True;
27             end
28         end
29         $tries \leftarrow tries + 1$;
30     end
31 end
```

Algorithm 2: Main service-chain allocation procedure.

```
1  Procedure AllocateChain($G, N^c, racks, cluster, \mathcal{G}^s, \alpha, \beta, \gamma$):
2      for $NodesSet$ in $\big\{\{\{n\} : n \in N^c\big\}, racks, clusters\}$ do
3          if AgPlaceChain($NodesSet, \alpha, \beta, \gamma, \mathcal{G}^s$) then
4              break;
5          end
6      end
7  end
```

in $rack1$ using the same procedure as followed in the accelerator-agnostic heuristic. It can be observed that the accelerator-aware VNF-PC heuristic results in using one less server-node as compared to the accelerator-agnostic heuristic for the deployment of VNF-chains s_1 and s_2.

Algorithm 3: VNF acceleration procedure

```
Procedure AccelVNF($f^s$, $n_p$, $node\_accels$, $t_s$):
    if atype($f^s$) $\notin A$ or $\mathcal{R}_{cpu}(n_p) < \big(cpu_0(f^s) - cpu_r(f^s)\big)$ or $\mathcal{R}_{pci}(n_p) < 2t_s$ then
        return False;
    else
        if atype($f^s$) $\notin node\_accels[n_p]$ then
            if r(atype($f^s$)) $> \mathcal{R}_{acc}(n_p)$ then
                return False;
            else
                $\mathcal{R}_{acc} \leftarrow \mathcal{R}_{acc} -$ r(atype($f^s$));
                $node\_accels[n] \leftarrow node\_accels[n] \cup \{$atype($f^s$)$\}$;
            end
        end
        $\mathcal{R}_{cpu}(n) \leftarrow \mathcal{R}_{cpu}(n) - \big(cpu_0(f^s) - cpu_r(f^s)\big)$;
        $\mathcal{R}_{pci}(n) \leftarrow \mathcal{R}_{pci}(n) - 2t_s$;
        return True;
    end
end
```

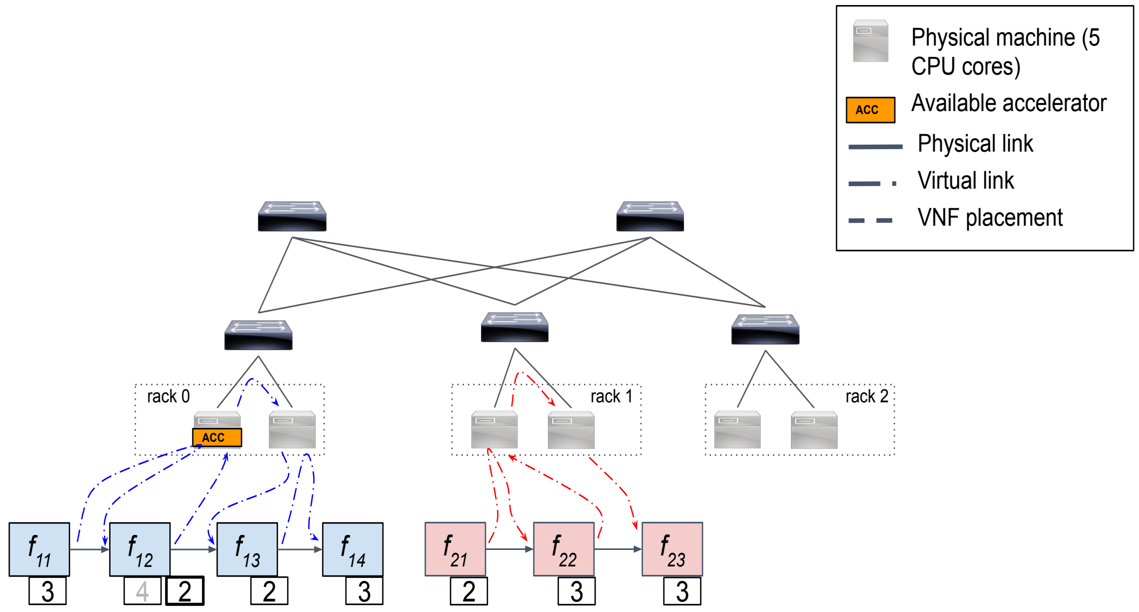

Figure 7: Illustration showing placement and chaining in accelerator-aware VNF-PC heuristic on the leaf-spine topology.

Algorithm 4: VNF chaining procedure

```
Procedure ChainVNFs($\alpha$, $\gamma$, $\mathcal{L}^s$, $G$):
    $G_0(N_0, E_0) \leftarrow G(N, E)$;
    $\gamma_0 \leftarrow \gamma$ ;
    for $(f^s_k, f^s_{k+1})$ in $\mathcal{L}^s$ do
        $done \leftarrow$ False;
        if $\alpha[f^s_k] \neq \alpha[f^s_{k+1}]$ then
            $P \leftarrow$ ShortestPaths($G$, $\alpha[f^s_k]$, $\alpha[f^s_{k+1}]$)*;
            for $p$ in $P$ do
                if bw($p$) $>= t_s$* then
                    for $(n_i, n_j)$ in $p$ do
                        $b(n_i, n_j) \leftarrow b(n_i, n_j) - t_s$*;
                        $\mathcal{R}_{bus}(n_i) \leftarrow \mathcal{R}_{bus}(n_i)$ - $2t_s$;
                        $\gamma[f^s_k, f^s_{k+1}] \leftarrow (n_i, n_j)$;
                        $done \leftarrow$ True;
                    end
                    break;
                end
            end
            if $done$ == False then
                $G(N, E) \leftarrow G_0(N_0, E_0)$;
                $\gamma \leftarrow \gamma_0$;
                return False;
            end
        end
    end
    return True;
end
```

Algorithm 5: Placement procedure for accelerate-able VNFs.

```
Procedure PlaceAccelVNFs($\alpha$, $\beta$, $\gamma$, $node\_accels$, $S$):
    $N_a \leftarrow \{n \in N^c : \mathcal{R}_{acc}(n) > 0\}$;
    for $s$ in $S$ do
        $F^s_{acc} \leftarrow \{f^s \in \mathcal{F}^s : atype(f^s) \in A\}$;
        for $f^s$ in $F_{acc}$ do
            $N_{f^s} \leftarrow \{n \in N^c : atype(f^s) \in node\_accels[n], 2t_s < \mathcal{R}_{bus}, \big(cpu_0(f^s) - cpu_r(f^s)\big) < \mathcal{R}_{cpu}(n)\}$
            if $N_{f^s} \neq \phi$ then
                $n_p \leftarrow$ select node from $N_{f^s}$ where previous VNF of the service chain $s$ was placed;
                $n_a \leftarrow \arg\min_n$ PathLen($n$, $n_p$) ;
            else
                $n_a \leftarrow$ select a node from $N_a$ with sufficient resources;
            end
            if AccelVNF($f^s$, $n_a$, $node\_accels$, $t_s$) then
                $\alpha[f^s], \beta[f^s] \leftarrow n_a, n_a$;
            end
        end
    end
    $used\_nodes \leftarrow$ all used nodes in $N$;
    return $used\_nodes$;
end
```

Algorithm 6: Accelerator-aware VNF-PC procedure.

```
Procedure AccelAwarePlaceChain($\alpha$, $\beta$, $\gamma$, $S$):
    $used\_nodes \leftarrow$ PlaceAccelVNFs($\alpha$, $\beta$, $S$);
    $S^p_{seg} \leftarrow \{\}$;
    for $n$ in $used\_nodes$ do
        $node\_chains \leftarrow \{s \in S : \exists f^s \in \mathcal{F}^s$ placed on $node\}$;
        $S^n_{seg} \leftarrow \{\}$;
        for $chain$ in $node\_chains$ do
            $S^n_{seg} \leftarrow S^n_{seg} \cup$ {all possible chain segments in $chain$};
        end
        /* place remaining segments of chains */
        for $seg$ in $S^n_{seg}$ do
            $\mathcal{G}^{seg} \leftarrow$ VNF forwarding sub-graph corresponding to $seg$;
            if $seg \not\subset S^p_{seg}$ then
                for $NodesSet$ in $\{\{node\}, rack\_node, cluster\_node\}$ do
                    if AgPlaceChain($\{NodesSet\}$, $\alpha$, $\beta$, $\gamma$, $\mathcal{G}^{seg}$) then
                        $f^{seg}_l \leftarrow$ leftmost VNF of $seg$;
                        $f^{acc}_l \leftarrow$ VNF which needs to be linked with the leftmost VNF of $seg$;
                        $f^{seg}_r \leftarrow$ rightmost VNF of $seg$;
                        $f^{acc}_r \leftarrow$ VNF which needs to be linked with the rightmost VNF of $seg$;
                        if ChainVNFs($\alpha$, $\gamma$, $\{(f^{acc}_l, f^{seg}_l)\}$, $G$) and ChainVNFs($\alpha$, $\gamma$, $\{(f^{seg}_r, f^{acc}_r)\}$, $G$) then
                            $S^p_{seg} \leftarrow S^p_{seg} \cup \{seg\}$;
                            break;
                        end
                    end
                end
            end
        end
    end
    $S_R \leftarrow S \setminus \{s \in S : \alpha[f^s] \neq \phi, \forall f^s \in \mathcal{F}^s\}$ ;
    /* placement and chaining of remaining service-chains */
    for $s$ in $S_R$ do
        for $NodesSet$ in $\{\{\{n\} : n \in N^c\}, racks, clusters\}$ do
            if AgPlaceChain($NodesSet, \alpha, \beta, \gamma, \mathcal{G}^s$) then
                break;
            end
        end
    end
end
```

9 Performance evaluation

The objective of this section is to assess the scalability and efficiency of the ILP model and VNF-PC heuristics using simulation experiments. We first describe the simulation environment used in our evaluation and then present the results obtained after performing experiments on the ILP model and heuristics.

9.1 Setup and Parameters

Table 3: Default values/range of various parameters involved in simulation experiments.

Parameter	Value or range	Parameter	Value or range		
$\mid S \mid$	[5, 250]	$c_o(f^c)$	3-5 (cores)		
$R_{cpu}(n)$	24 (cores)	$c_i(f^c)$	(0.40 - 0.60)$c_o(f^c)$		
$R_{acc}(n)$	0,100k (LUTs)	ρ^{vnf}_{acc}, ρ^{n}_{acc}	0.20, 0.30		
$R_{bus}(n)$	80 (Gbps)	$b(n_i, n_j)$	10, 40 (Gbps)		
VNF-chain length	4-6	accel. type (a)	a_1	a_2	a_3
t_s	100-500 (Mbps)	$r(a)$ (LUTs)	40k	28k	30k
c_n	1, 1.20				

The ILP model for VNF-AAP problem has been built using Python API of IBM's ILOG CPLEX called DOcplex (Decision Optimization CPLEX Modeling). DOcplex provides a user-friendly API to write the ILP model which is then solved by the CPLEX solver. All heuristic algorithms are written in Python programming language. We used an Intel Xeon server machine with quad-core CPU @ 2.40GHz with 16GB of RAM memory running Ubuntu-16.04 OS to carry out evaluations of the ILP and heuristics. Each data point reported in the evaluations indicates an average over 10 iterations along with the corresponding confidence interval of one standard deviation (68%).
For evaluation of heuristics, we have considered two different DC topologies for simulating the physical network: (i) three-tier and (ii) leaf-spine. For three-tier topology, we vary the value of k to adjust the size of the network. For example e.g. when k=6 we will have k=6 pods, each pod containing $k/2$ =3 access switches and $k/2$ = 3 aggregate switches. Each access-switch (ToR switch) is connected to 6/2=3 server-nodes and therefore the total number of server-nodes in all the pods equal to 54. For leaf-spine, we have considered 4 core-switches and 16 leaf-switches (ToR switch). Each leaf-switch is connected to 16 server-nodes, therefore, resulting in a total of 320 server-nodes. In both the topologies, the links connecting server-nodes with ToR switches and switches with switches are 10Gbps and 40Gbps links respectively.
Each server-node has 24 CPU cores, 16GB/s of PCIe bandwidth, and has 100k LUTs if a hardware-accelerator card is attached to the server-node. For simplicity we assume cost c_n of a server-node to be 1.20$ if it is attached with a hardware-accelerator, otherwise 1$. The other parameters considered in evaluations are given in Table 3.

9.1.1 Comparison of ILP and Heuristic

Before presenting evaluation results regarding total node costs, we first report total execution times for ILP and heuristic approach. Fig. 9 shows distribution of total execution for both approaches when deploying 5 VNF-chains on a leaf-spine topology shown in Fig. 8. As expected, it can be observed that the execution time of ILP-approach is orders of magnitude larger than the heuristic approach. Moreover, when the number of VNF-chains to be deployed on the given topology becomes large $\mid S \mid \geq 15$ the total execution time could reach up to several hours.
Fig. 10 shows the evolution of CPLEX solutions with time for the deployment of 15 VNF-chains. It can be observed that CPLEX takes about 2 hours to complete the execution for this instance, yet the gap between the incumbent solution and the lower-bound estimated by CPLEX after one hour is negligible (0.5%). Nevertheless, only small sizes instances of the VNF-AAPC problem can be solved using the ILP approach in a reasonable time.
Fig. 11 gives the comparison between ILP and heuristic in terms of total nodes cost for the deployment of different number of VNF-chains. Here we have limited the maximum execution time of CPEX instances to one hour. The bar chart shows (i) ILP incumbent solution (ILP) and (ii) best lower-bound (ILP-LB) estimated by CPLEX until one hour and (iii) VNF-AAPC heuristic solution. We can observe that there exists a small penalty (on average $\sim$5%) when using the heuristic approach instead of the ILP approach. As mentioned earlier, the gap between ILP-LB and ILP is almost negligible after one hour of CPLEX execution time.
For the deployment of 20 VNF chains, we can observe that the total nodes cost using the ILP approach is higher

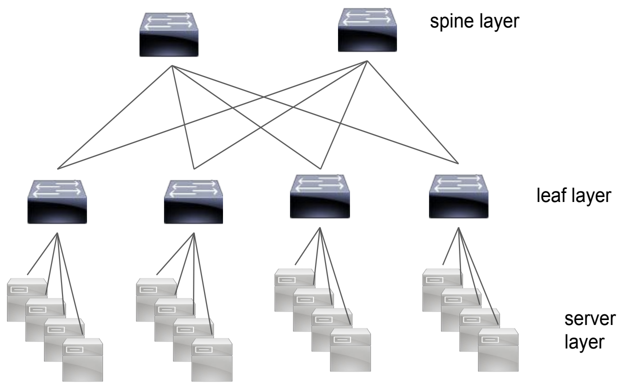

Figure 8: Leaf-spine topology used for the evaluation of ILP and heuristic.

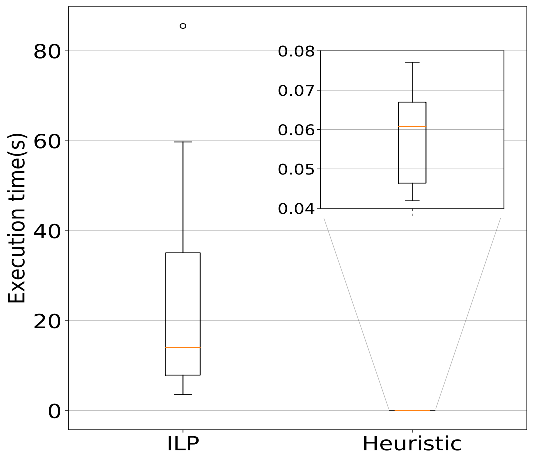

Figure 9: Comparison of ILP model and heuristic in terms of total execution times for the leaf-spine topology.

than with the heuristic method. As the total time of execution is limited, CPLEX was not able to reach the optimal solution in the given time and the VNF-AAPC heuristic method is able to achieve more efficient allocation than the ILP approach. Although, CPLEX can find the optimal solution if allowed to run without any time limitation, the performance of the heuristic is still very close to the estimated lower bound by CPLEX. As mentioned earlier, it will be impracticable to use ILP to solve problem instances of size larger than 15 VNF-chains.

9.1.2 VNF-PC Heuristic Comparison

Here, we compare the performance of heuristic algorithms among themselves in terms of the following performance metrics.

1. First, total node cost is the cost due to server-nodes which also includes additional costs due to installation of hardware-accelerators in some of the server nodes. The comparison of node costs will indicate the resulting cost-saving in NFVi by using a particular VNF-PC scheme.

2. Second, β/α is the ratio of total VNFs allocated hardware-accelerators to the total VNFs in all VNF-chains. It is possible that VNF-PC algorithm might not allocate an accelerator to an accelerate-able VNF. This metric shows the efficiency of the VNF-PC algorithm in terms of utilization of hardware-accelerator resources. A higher value of β/α indicates efficient allocation of hardware-accelerator resources by the VNF-PC algorithm.

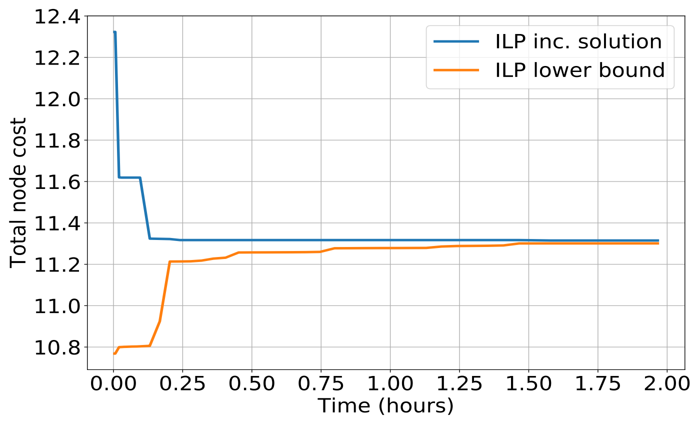

Figure 10: Evolution of ILP's incumbent solution and lower-bound for (a) full execution of CPLEX and (b) for first one hour.

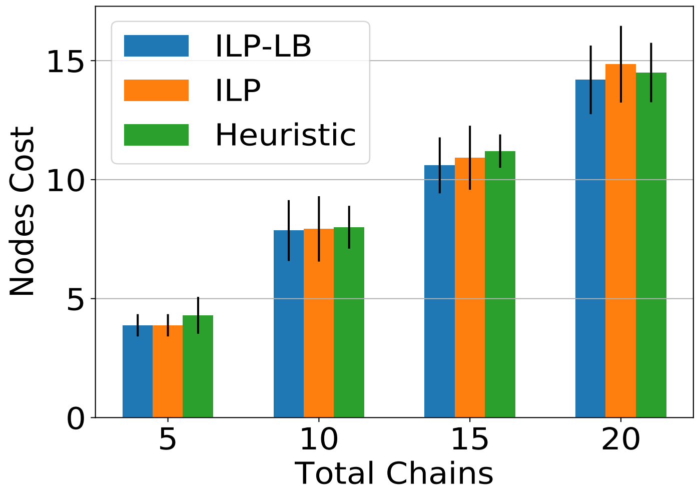

Figure 11: Comparison of ILP model and heuristic in terms of total node costs in the leaf-spine topology for different number of VNF-chains.

3. Third, CPU_{rem} is the average amount of CPU cores remaining per server-node left unallocated after the completion of VNF-PC. A high CPU_{rem} indicates the poor consolidation of VNFs, thereby resulting in overall inefficient allocation of resources.

To bench-mark the performance of the proposed VNF-AAPC heuristic, we also evaluate the performance of the accelerator-agnostic VNF-PC heuristic. The accelerator-agnostic VNF-PC heuristic will serve as the baseline for the evaluation of our VNF-AAPC heuristic.
Fig. 12 (a) and (d) show the total node costs incurred to the operator as a result of deployment of different numbers of VNF-chains on three-tier and leaf-spine DC topologies, respectively. In both topologies, the results show the lowest resource cost in case of accelerator-aware VNF-PC heuristic. This arises from the efficient consolidation of VNFs, as explained below, by the accelerator-aware heuristic as contrast to the poor VNF consolidation in accelerator-agnostic heuristic.
For the accelerator-agnostic VNF-PC, the VNF placement process is unaware of the presence of accelerator resources on a server-node. The chance of an accelerator being allocated to a VNF depends on the odds of an accelerate-able VNF being placed on a server-node with a hardware-accelerator. The probability (p_{acc}) of allocation of an accelerator to a VNF using the accelerator-agnostic VNF-PC heuristic is thus given by the product $p_{acc} = \rho_{acc}^{vnf} \rho_{acc}^{n}$. Here, ρ_{acc}^{vnf} is the fraction of VNF that can be offloaded using a hardware-accelerator (accelerate-able VNFs) and

ρ_{acc}^{n} is the fraction of server-nodes attached with a hardware-accelerator. Therefore, p_{acc} gives odds of accelerator allocation to a VNF with the accelerator-agnostic heuristic. This can also be verified from the resulting β/α ratios depicted in Fig. 12 (b) and (e). The β/α ratio for the accelerator-agnostic heuristic remains smaller than the accelerator-aware heuristic for any value of total VNF-chains. The explicit allocation of accelerators to VNFs occurs in the accelerator-aware heuristic which is in contrast to the accelerator-agnostic heuristic where accelerator-allocation is arbitrary. Moreover, we observed an increase in beta/alpha ratio with the increasing number of total VNF-chains. This observation can be attributed to the fact that the accelerator-aware heuristic attempts to reuse the deployed accelerator instances and nodes attached with hardware-accelerators. Therefore, increasing the number of VNF-chains causes an increase in the number of candidates for accelerator allocation; thus resulting in a better beta/alpha ratio.

CPU_{rem} metrics for both heuristics are depicted in Fig. 12 (c) and (f). VNF consolidation on server-nodes tends to increase with the total number of VNF chains as the chance of placing VNF on a server-node will increase with the increase in the total number of VNFs. This is confirmed by the decreasing CPU_{rem} with the increasing number of VNF-chains for both topologies. Moreover, as more VNFs are granted accelerator using the accelerator-aware heuristic compared to the accelerator-agnostic heuristic, the corresponding CPU_{rem} is smaller and therefore more VNF consolidation is achieved.

We also try to show the impact of changing the fraction of nodes ρ_{acc}^{n} with an attached hardware-accelerator card on overall performance metrics; when deploying the same set of VNF-chains. For this experiment, we decreased the fraction of server-nodes attached with a hardware-accelerators in a three-tier topology with $k = 10$ and measured the performance metrics for both heuristics. It can be observed from Fig. 13 that the total nodes cost for accelerator-aware heuristic remains less than that of the accelerator-agnostic heuristic for all values of ρ_{acc}^{n}. Also, as the fraction of nodes with hardware-accelerator ρ_{acc}^{n} is reduced, the additional accelerator cost is decreased for both accelerator-agnostic and accelerator-aware VNF-PC heuristics which is negated by the additional costs due to the requirement of extra server-nodes.

As expected, β/α ratio decreases with decreasing ρ_{acc}^{n} for both accelerator-agnostic and accelerator-aware heuristics. The β/α ratio decreases almost linearly with the decrease in ρ_{acc}^{n}. This, again, arises from the fact that the probability of accelerator allocation in the accelerator-agnostic heuristic is directly proportional to ρ_{acc}^{n} value which is not the case with the accelerator-aware heuristic. As placement decisions for accelerate-able VNFs are separate in accelerator-aware VNF-PC heuristic, there is no drastic impact on its β/α ratio with a decrease in ρ_{acc}^{n} value. There isn't any significant change in CPU_{rem} for both heuristics with the change in ρ_{acc}^{n} values. However, VNF consolidation for accelerator-aware heuristic is better than the accelerator-agnostic heuristic as was expected.

9.2 Overall cost analysis

In this section, we analyze the cost-saving achieved as a result of incorporating hardware-acceleration in NFVi. We assume the total number of server-nodes (without any hardware-accelerator) required for the deployment of a given set of VNF-chains is N. The total cost $Cost_0$ incurred to the operator as a result of running server-nodes can be expressed as follows:

$$Cost_0 = Nc_0 \tag{14}$$

Here, c_0 is the cost of running a single server-node (without hardware-accelerator) and N is the total number of server-nodes required for the deployment of a set of VNF-chains.

After the installation of hardware-accelerators in server-nodes, the total cost of deployment $Cost_{acc}$ ~~of~~ for the same set of VNF-chains can be expressed as follows:

$$Cost_{acc} = (1 + c_{acc})c_0N(1 - \rho_{red})\rho_{acc}^{n} + c_0N(1 - \rho_{red})(1 - \rho_{acc}^{n}) \tag{15}$$

The first term and the second term of eq. 15 refer to the cost of using server-nodes attached with and without hardware-accelerators, respectively. c_{acc} is the additional cost of installation of hardware-accelerator in a server-node relative to the original server-node cost, ρ_{acc}^{n} is the fraction of server-nodes that are installed with a hardware-accelerator and ρ_{red} is the relative (total number of server-nodes) reduction in the number of server-nodes after hardware-acceleration for VNFs.

Fig. 14 compares the total server-nodes required for the deployment of 100 VNF-chains on a leaf-spine topology in two cases, (i) when server-nodes are not attached with any hardware accelerator card and (ii) when sever-nodes are attached with a hardware-accelerator card. We have used the accelerator-agnostic heuristic for the case when NFVi does not contain any hardware-accelerators and for the case when NFVi contains hardware-accelerators, we have used the accelerator-aware heuristic. It can be observed that the relative reduction ρ_{red} in the total number

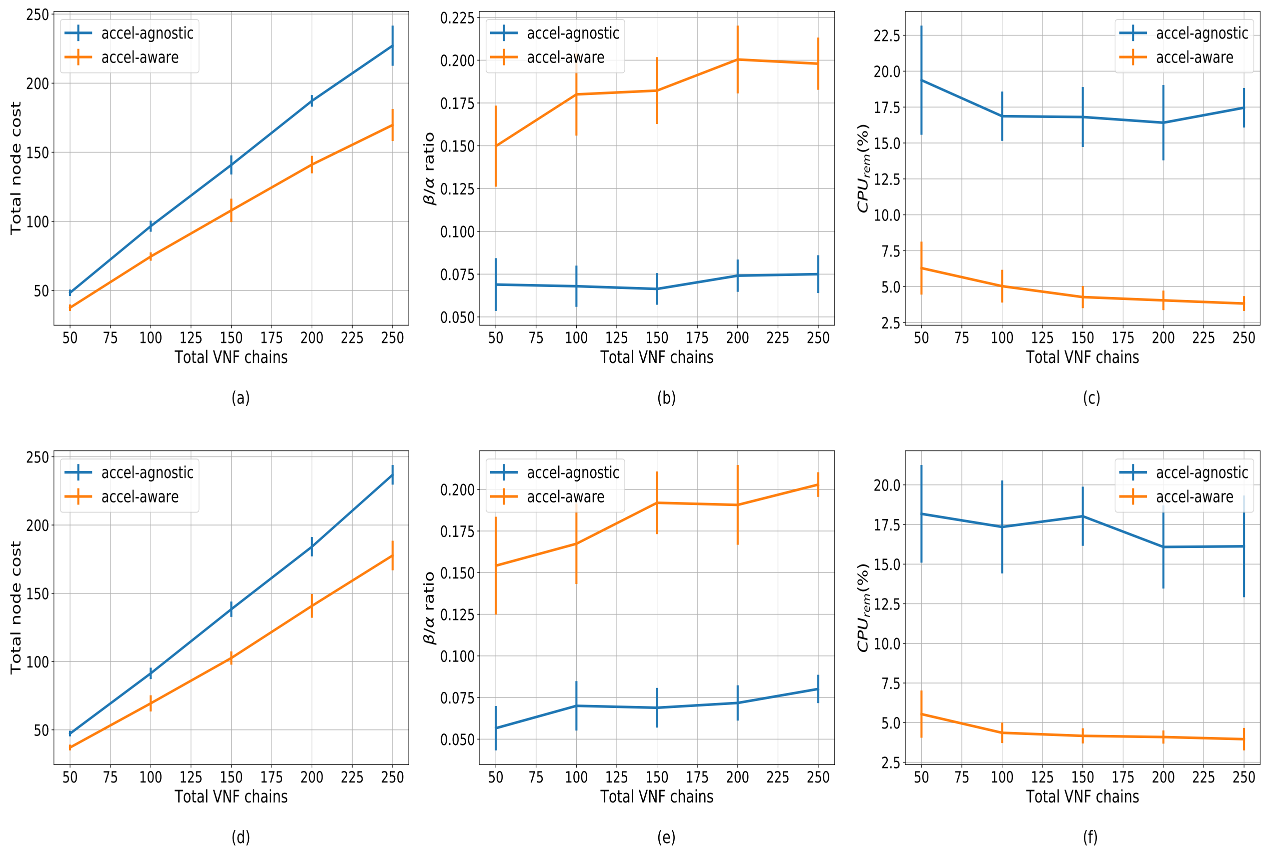

Figure 12: Comparison of accelerator-agnostic and accelerator-aware heuristics in terms of total node costs, β/α ratio and CPU_{rem} for three-tier and leaf-spine topologies. Plots for three-tier topology are shown in (a), (b) and (c) and plots in (d), (e) and (f) correspond to leaf-spine topology.

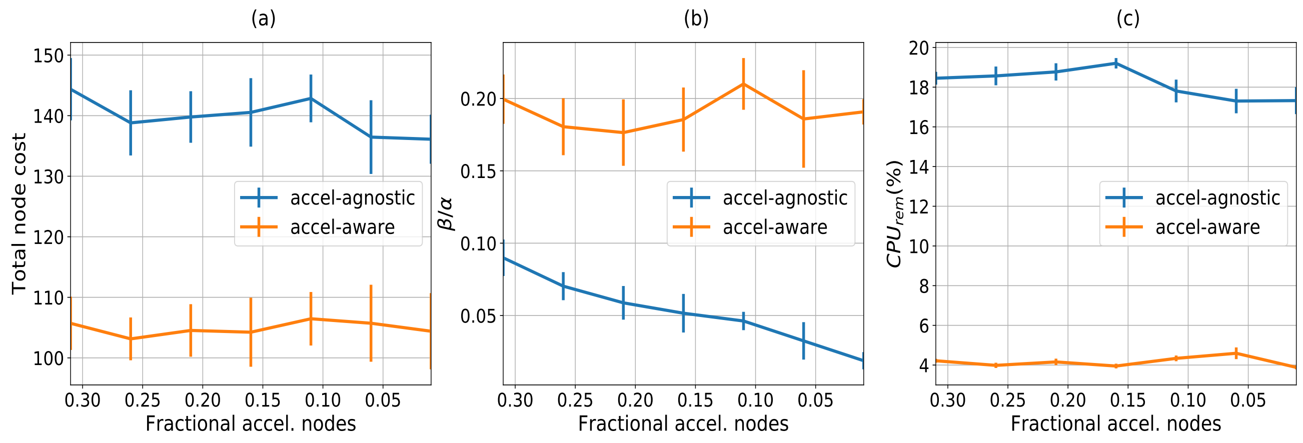

Figure 13: Impact of fraction of nodes with accelerator ρ^n_{acc} on the total number of required nodes for deployment of 100 VNF-chains on three-tier topology ($k = 10$) using accelerator-agnostic and accelerator-aware VNF-PC heuristics.

of server-nodes by using hardware-accelerators is 18-20%.

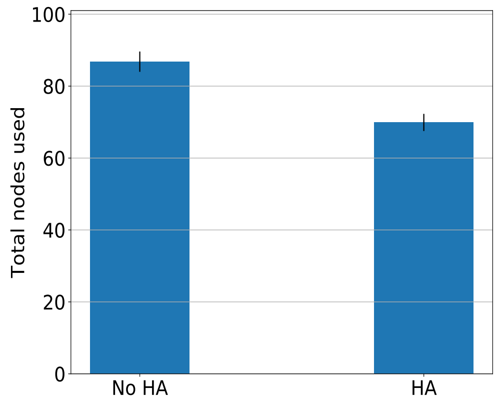

Figure 14: Total server-nodes required for the deployment of 100 VNF-chains on a leaf-spine topology in two cases, (i) when server-nodes are not attached with any hardware accelerator card and (ii) when sever-nodes are attached with a hardware-accelerator card.

Relative cost-savings (G) is the relative reduction in the total cost as a result of using hardware-acceleration in NFVi. G can be obtained by using the expression shown below:

$$G = (Cost_0 - Cost_{acc})/Cost_0 = 1 - (1 - \rho_{red}).\big(1 - \rho_{acc}^n + (1 + c_{acc}).\rho_{acc}^n\big) \tag{16}$$

Eq. 16 gives an expression for the achievable cost-saving in terms of relative server-node reduction ρ_{red} and additional costs of hardware-accelerators (c_{acc}). Using eq. 16, we can plot the required minimum ρ_{red} to achieve a given cost-saving G as shown in Fig. 15. Fig. 15 shows four different contours corresponding to four different G values. For example, to achieve an overall 15% savings ($G = 0.15$) on server-nodes cost, VNF-PC algorithm should achieve at least 18% reduction of total server-nodes, when an additional cost of 18.5% is needed for the installation of hardware-accelerators. As expected, one can observe that higher G values require high server-node reduction ρ_{red} and low additional costs c_{acc}. Therefore, efficient accelerator-aware VNF-PC heuristics are required to gain the benefits of hardware-accelerator even when costs of hardware-accelerators are expected to reduce in the future. As stated earlier, about $\rho_{red} = 18 - 20\%$ reduction in total server-nodes can be obtained using our VNF-AAPC heuristic. As a result, about 15% of overall cost-saving (G) is achievable by the operator.

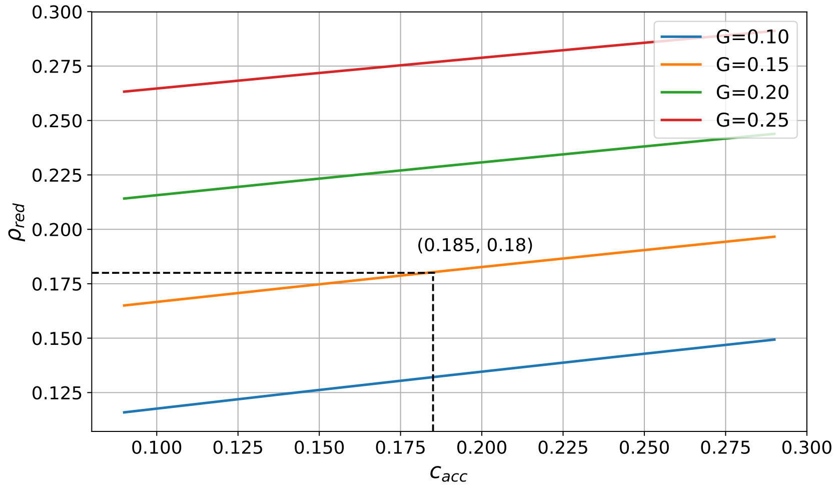

Figure 15: Variation of relative node reduction ρ_{red} with respect to additional hardware-accelerator cost for VNF-PC heuristic's. Each line represent a locus of all points with fixed value of cost-saving G.

10 Conclusion

NFVi generally includes all hardware and software components required to build a virtualized environment for running VNFs. However, due to specific performance or energy goals, it becomes essential to provide some kind of acceleration to certain VNFs. However, the current NFVi resource allocation models do not consider hardware-accelerator resources while performing placement and chaining of VNFs; therefore, resulting in an inefficient utilization of NFVi resources.
In this paper, we modeled the VNF-AAPC problem for NFV environments containing hardware-accelerators along with the usual NFVi resources. To tackle the VNF-AAPC problem, we proposed two approaches: (i) ILP method and (ii) heuristic algorithm. As opposed to the ILP-approach, the heuristic-based method is able to scale with the problem size at the cost of a small penalty. Both approaches aim at minimizing the cost incurred to the operator due to the utilization of resources for the deployment VNF-chains. The heuristic-based approach performs tasks of VNF placement and chaining in two different phases: (i) Placement of accelerate-able VNFs, (ii) Placement and Chaining of remaining VNF-chain segments. The proposed methods were also evaluated using simulation experiments and then were compared in terms of their resulting cost and other performance metrics. The simulation results indicate that the accelerator-aware heuristic approach can achieve 12-14% cost-savings as compared to the accelerator-agnostic heuristic. Finally, we also performed overall cost-analysis on the use of hardware-accelerators in NFV environments. The analysis shows that the proposed accelerator-aware VNF-PC heuristic could be used to achieve significant cost-savings when using hardware-accelerators in NFVi.
Hardware-accelerators are not only utilized in cloud DCs for performance enhancements of VNFs but also in other scenarios e.g. network edges, Centralized Radio Access Networks (CRANs). To reduce the energy cost and meet strict performance requirements in CRAN, various techniques to offload baseband processing functions, e.g. iFFT/FFT, turbo-coding, using hardware-accelerators are being investigated. However, the problem to model resource dimensioning for virtual base stations in cloud RANs (C-RAN) architectures with hardware-accelerators still remains to be investigated.

11 Acknowledgments

This work was funded through NGPaaS, under the grant number 761557, in the scope of the European Commission Horizon 2020 and 5G-PPP programs.

References

[1] B. Yi, X. Wang, K. Li, M. Huang, *et al.*, "A comprehensive survey of network function virtualization," *Computer Networks*, vol. 133, pp. 212–262, 2018.

[2] "Architectural framework," 2013. Online; Release v1.1.1 2013-10.

[3] L. Linguaglossa, S. Lange, S. Pontarelli, G. Rétvári, D. Rossi, T. Zinner, R. Bifulco, M. Jarschel, and G. Bianchi, "Survey of performance acceleration techniques for network function virtualization," *Proceedings of the IEEE*, vol. 107, no. 4, pp. 746–764, 2019.

[4] N. Nikaein, "Processing radio access network functions in the cloud: Critical issues and modeling," in *Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services*, pp. 36–43, ACM, 2015.

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

[6] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[7] S. Gebert, A. Müssig, S. Lange, T. Zinner, N. Gray, and P. Tran-Gia, "Processing time comparison of a hardware-based firewall and its virtualized counterpart," in *International Conference on Mobile Networks and Management*, pp. 220–228, Springer, 2016.

[8] "Addressing 5G Network Function requirements." White Paper, 2018. Online.

[9] G. P. Sharma, W. Tavernier, D. Colle, and M. Pickavet, "Dynamic hardware-acceleration of VNFs in NFV environments," in *2019 Sixth International Conference on Software Defined Systems (SDS)*, pp. 254–259, June 2019.

[10] Z. Martinasek, J. Hajny, D. Smekal, L. Malina, D. Matousek, M. Kekely, and N. Mentens, "200 gbps hardware accelerated encryption system for fpga network cards," in *Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security*, pp. 11–17, ACM, 2018.

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

[12] X. Ge, Y. Liu, D. H. Du, L. Zhang, H. Guan, J. Chen, Y. Zhao, and X. Hu, "OpenANFV: Accelerating network function virtualization with a consolidated framework in openstack," in *ACM SIGCOMM Computer Communication Review*, vol. 44, pp. 353–354, ACM, 2014.

[13] A. Albanese, P. S. Crosta, C. Meani, and P. Paglierani, "Gpu-accelerated video transcoding unit for multi-access edge computing scenarios," in *Proceeding of ICN*, 2017.

[14] M. Masoudi, M. G. Khafagy, A. Conte, A. El-Amine, B. Françoise, C. Nadjahi, F. E. Salem, W. Labidi, A. Süral, A. Gati, *et al.*, "Green mobile networks for 5g and beyond," *IEEE Access*, vol. 7, pp. 107270–107299, 2019.

[15] S. Han, K. Jang, K. Park, and S. Moon, "Packetshader: a gpu-accelerated software router," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 195–206, 2011.

[16] X. Yi, J. Duan, and C. Wu, "Gpunfv: a gpu-accelerated nfv system," in *Proceedings of the First Asia-Pacific Workshop on Networking*, pp. 85–91, 2017.

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

[18] Z. Bronstein, E. Roch, J. Xia, and A. Molkho, "Uniform handling and abstraction of NFV hardware accelerators," *IEEE Network*, vol. 29, no. 3, pp. 22–29, 2015.

[19] Y. Watanabe, Y. Kobayashi, T. Takenaka, T. Hosomi, and Y. Nakamura, "Accelerating NFV application using cpu-fpga tightly coupled architecture," in *2017 International Conference on Field Programmable Technology (ICFPT)*, pp. 136–143, IEEE, 2017.

[20] "Acceleration technologies; report on acceleration technologies & use cases," 2015. Online; Release v1.1.1 2015-12.

[21] H. Moens and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *10th International Conference on Network and Service Management (CNSM) and Workshop*, pp. 418–423, IEEE, 2014.

[22] 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)*, pp. 7–13, Oct 2014.

[23] 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)*, pp. 171–177, IEEE, 2015.

[24] 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)*, pp. 255–260, IEEE, 2015.

[25] S. Sahhaf, W. Tavernier, M. Rost, S. Schmid, D. Colle, M. Pickavet, and P. Demeester, "Network service chaining with optimized network function embedding supporting service decompositions," *Computer Networks*, vol. 93, pp. 492–505, 2015.

[26] F. Carpio, S. Dhahri, and A. Jukan, "VNF placement with replication for load balancing in NFV networks," in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2017.

[27] H. Fan, Y. Hu, S. Zhang, and Q. Ren, "Hardware acceleration resource allocation mechanism for VNF," *Procedia computer science*, vol. 131, pp. 746–755, 2018.

[28] S. Dräxler and H. Karl, "SPRING: Scaling, placement, and routing of heterogeneous services with flexible structures," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, pp. 115–123, June 2019.

[29] G. P. Sharma, W. Tavernier, D. Colle, and M. Pickavet, "VNF-AAP: Accelerator-aware virtual network function placement," 2019.

[30] N. Kodirov, S. Bayless, F. Ruffy, I. Beschastnikh, H. H. Hoos, and A. J. Hu, "VNF chain allocation and management at data center scale," in *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems*, pp. 125–140, ACM, 2018.