Performance-Aware Orchestration of P4-based Heterogeneous Cloud Environments

Hasanin Harkous 1, Mu He 2, Rastin Pries 2, Michael Jarschel 2, wolfgang kellerer 2, and Bassel Aboul Hosn 2

1Noki
2Affiliation not available

October 31, 2023

Abstract

The recent trend to deploy programmable packet processors in cloud environments enhances the packet processing capability without losing the flexibility to adapt the functions at runtime. In particular, distributed edge clouds can have a heterogeneous programmable processing substrate made up of different classes of devices: CPUs, NPUs, FPGAs, etc. However, managing the allocation of workloads in such a heterogeneous programmable processing substrate, in particular deciding where to instantiate a certain function, is a non-trivial task with many decisive functional and QoS-related factors.

In this paper, we propose a mathematical model for optimizing the embedding of Service Function Chains implemented in P4, while considering the functional and QoS requirements associated with embedding requests, and the various types of processing devices that have different properties in terms of processing delay and supported features. To satisfy delay requirements, the problem formulation utilizes performance models to predict the forwarding latency associated with different candidate embedding options. Furthermore, a greedy solution is proposed to solve the problem in an efficient manner. Finally, a detailed numerical evaluation is conducted to evaluate the formulated model when different workload and infrastructure characteristics are varied and to evaluate the effectiveness of the proposed greedy solution.
Performance-Aware Orchestration of P4-based Heterogeneous Cloud Environments

Hasanin Harkous, Bassel Aboul Hosn, Mu He, Michael Jarschel, Rastin Pries, Wolfgang Kellerer

Abstract—The recent trend to deploy programmable packet processors in cloud environments enhances the packet processing capability without losing the flexibility to adapt the functions at runtime. In particular, distributed edge clouds can have a heterogeneous programmable processing substrate made up of different classes of devices: CPUs, NPUs, FPGAs, etc. However, managing the allocation of workloads in such a heterogeneous programmable processing substrate, in particular deciding where to instantiate a certain function, is a non-trivial task with many decisive functional and QoS-related factors.

In this paper, we propose a mathematical model for optimizing the embedding of Service Function Chains implemented in P4, while considering the functional and QoS requirements associated with embedding requests, and the various types of processing devices that have different properties in terms of processing delay and supported features. To satisfy delay requirements, the problem formulation utilizes performance models to predict the forwarding latency associated with different candidate embedding options. Furthermore, a greedy solution is proposed to solve the problem in an efficient manner. Finally, a detailed numerical evaluation is conducted to evaluate the formulated model when different workload and infrastructure characteristics are varied and to evaluate the effectiveness of the proposed greedy solution.

Index Terms—SFC Embedding, Orchestration, Cloud Environments, Programmable Data Plane, P4, NFV, SDN.

I. INTRODUCTION

The adoption of edge clouds enables the offloading of processing workloads from end systems to physical locations in the near vicinity of users. This technology requires building flexible networks that can be reconfigured easily to satisfy different functionality demands, without compromising the required performance levels. Moreover, unlike data centers, which are usually designed with maximum homogeneity in terms of processing substrate, the edge clouds are inherently distributed and full of heterogeneity [1]. The emergence of programmable packet processors such as SmartNICs and Infrastructure Processing Units (IPUs) provides a viable solution that can be used as a substrate for these cloud environments that can deliver high flexibility without compromising on performance [2]. Using the P4 language [3] for programming these packet processors is a promising option because of its wide support from both the research and industrial communities [4]. Moreover, being target-independent, P4 offers a viable solution for programming the heterogeneous edge clouds as P4 programs, with minor tailoring based on the supported P4 architecture, can be used to program different types of packet processors such as software switches, NPUs, FPGAs, or ASIC switches. This offers great flexibility and agility in running such heterogeneous processing environments. We call the flexible NFV processing environment that supports both hardware and software-based processing using P4 programmability a P4-enhanced NFV environment.

However, different P4 packet processors have different functional capabilities and limitations in terms of supported operations and function blocks. More interestingly, these devices have different performance levels as revealed in [5], where measurements showed that there is a large difference in the forwarding latency of the different state-of-the-art P4 targets. For instance, when running the same Layer3 Forwarding P4 program on different P4 devices, the forwarding latency could vary by more than 12 folds. Additionally, these measurements also revealed that the processing latency could increase by up to 85% when the complexity of the P4 program increases even when using the same P4 device.

To this end, a smart solution is needed to orchestrate the placement of processing workloads into heterogeneous NFV environments made up of distinct packet processors while satisfying the different functional and QoS requirements associated with the given workload.

In an earlier work [6], we formulated and evaluated an optimization problem that deals with the planning of the infrastructure substrate of P4-enhanced NFV environments. The optimization problem looks for the optimal set of P4 packet processors that can handle a given Network Function (NF) processing workload and the placement solution of this workload into the selected hosting devices. The different requirements of the NFs workload and the distinct performance and capabilities of the available P4 devices in the cloud infrastructure are taken into consideration while formulating the problem. The objective function is to maximize the forwarding performance in the system while minimizing the capital expenditure costs when selecting the optimal set of P4 devices that can handle the given processing workload.

This work extends [6] by considering the runtime orchestration and management of the P4-enhanced NFV environments. While our early work in [6] is concerned with the offline optimal planning for selecting the network’s substrate and placing NFs, the problem studied in this work is concerned with optimizing the placement of Service Function Chains (SFCs), i.e., NFs and logical links, into an already given built-up network with certain topology. The objective of the placement decision is to minimize the operational cost instead of the capital expenditure as the infrastructure substrate is already selected. Moreover, calculating the resultant delay of placing SFCs into the network needs more careful handling compared to doing this calculation for NFs placement.

The main contributions of this paper are the following:

• We formulate an optimization problem that targets finding
the optimal embedding (i.e., placement and routing) of SFCs into a given P4-enhanced NFV environment at runtime. The problem formulation guarantees to satisfy the functional and QoS requirements associated with different SFC embedding requests.

- We integrate our pre-developed performance models in [4] and [5] into the problem formulation to predict a priori the forwarding latency that results from different embedding options: placing any NF on any P4 device. This performance-awareness feature enables us to orchestrate the P4-enhanced NFV environments in a highly efficient way, wherein we can satisfy the required delay requirements associated with SFC embedding requests.

- We design and implement a greedy solution to solve the problem efficiently.

- We conduct a detailed numerical evaluation to evaluate the formulated model when different workload and infrastructure characteristics are varied and to evaluate the effectiveness of the proposed greedy solution.

The rest of this paper is organized as follows. In Section II, we define the context and objective of the studied optimization problem, as well as its distinction from state-of-the-art literature. In Section III, we define the mathematical formulation of the studied optimization problem. A greedy solution is proposed in Section IV. The evaluation of the optimization problem and the greedy algorithm is presented in Section V. Finally, Section VI concludes the paper.

II. PROBLEM CONTEXT AND RELATED WORK

In Subsection II-A, we describe the scenario related to the considered optimization problem in this paper. Then, we distinguish the scope of this problem from state-of-the-art works in Subsection II-B.

A. Problem Context

The recent trend of deploying programmable packet processors in cloud environments improves packet processing capability without sacrificing the ability to adapt functions at runtime [6]. However, managing network functions, particularly deciding where to instantiate a specific function, is a difficult task with numerous deciding factors. First, the orchestrator should take into consideration that the NFs to be managed in the environment have different functional and performance requirements. For example, a NF can be compatible with specific P4 architecture that supports programming some function blocks, or it can require certain acceleration functions, or it can demand some QoS levels in terms of delay or throughput, etc. Second, the P4-enhanced NFV environment is made up of a heterogeneous pool of P4 packet processors with different capabilities and performance levels. For example, each P4 packet processor may support a different P4 architecture or may include certain acceleration functions, or have different performance levels as analyzed in [5].

To this end, we formulate and solve the Performance Aware P4 SFC Embedding (PA-P4SFC-E) problem that targets optimizing the deployment of SFCs that have distinct requirements into a P4-enhanced NFV environment made up of different P4 packet processors with distinct capabilities. The P4 technology is selected in this work because of its wide adoption by different programmable packet processors, and its target-independence feature that permits using it to program these packet processors although they belong to different processing platforms.

Fig. 1 depicts the considered scenario. The workload to be processed by the environment is presented as a set of SFCs. Each SFC is a connected sequence of P4-based NFs. If there is branching in the SFC definition, we consider each path of the acyclic graph as one sequential SFC. Each NF has different functional requirements such as compatible P4 architecture or required acceleration functions (expressed in terms of P4 externs), etc. Moreover, each NF describes a different processing pipeline written as a different P4 program with a different set of constituting P4 atomic constructs or operations. Besides the requirements associated with each constituting NF, each SFC also has requirements on the throughput and delay that need to be satisfied as shown in Fig. 1. The model also checks for the possibility of NF sharing among the requested set of SFCs to be placed to avoid duplicate packet processing and save processing resources.

The problem formulation also recognizes that different P4 packet processors have different functional capabilities and limitations, as well as different performance levels. The optimization problem constrains the placement of the SFC workload into P4 packet processors based on the requirements of NFs and the capabilities of the candidate P4 packet processors. These constraints ensure the compatibility between the NF and the hosting P4 device. The checked constraints include and are not limited to the compatibility of P4 architecture, availability of acceleration functions, availability of memory resources in terms of the total number of rules that can be placed, capacity in terms of throughput, and the maximum number of atomic P4 constructs that can be processed in a device.

Although the formulated model can be solved for any topology, a simple multi-rack topology is assumed in this scenario as depicted in Fig. 1. The infrastructure is made up of racks of servers with CPU processors. Each server can be equipped with an NPU-based SmartNIC or an FPGA-based SmartNIC. The servers and the two types of SmartNICs can
support P4 programmability. The racks are connected via Topof-Rack (ToR) switches. These ToR switches could be P4programmable ASIC switches or traditional switches that only perform L2Fwd functionality. Finally, the interconnection of the racks is achieved via traditional L2Fwd switches. The abstract representation of this network is also shown in Fig. 1, where different P4 packet processors are presented as nodes with different colors. The links between the SmartNICs and the CPU-based servers abstract the PCIe bus connectivity between these devices when SmartNICs are plugged into the servers. On the other end, the connections between the SmartNICs and the ToR switches present the case when the physical interfaces of the SmartNICs are connected to the ToR switches. For the sake of generality, the scenario recognizes the cases when traditional non-programmable switches are used in the infrastructure for interconnection. We assume that all P4 packet processors can process all their incoming traffic at line rate, so only the capacity of the connecting links (and not the devices) is considered in throughput constraints. It is also assumed that all P4 programmable devices in the network can serve as ingress or egress nodes for the traffic processed in the network. So each node is assumed to have another pair of ports that can serve as ingress and egress interfaces for traffic to enter or leave the network. The throughput constraint is also applied for these ingress and egress ports according to the throughput capacity of each P4 packet processor.

The problem in this scenario is to search for the optimal embedding of a workload of SFCs into a given P4 infrastructure, while satisfying all the functional and QoS constraints associated with each SFC. The component NFs of each SFC should be properly placed into P4 devices while recognizing the requirements of the NFs and the capabilities of the P4 devices. Moreover, the routing between the NFs constituting an SFC should also be provisioned. The pre-developed performance models from literature [5] are utilized in the formulated problem to perform a priori calculations of the delays associated with different embedding options. The applied method takes into account the characteristics of cutting-edge P4 devices at the granularity of the execution of various P4 atomic constructs. Each NF component of the SFC to be embedded is decomposed into a set of atomic P4 constructs in order to find the best set of P4 devices that can host that SFC such that the required delay QoS levels are satisfied. The objective function in this scenario targets minimizing the operational cost in the system by minimizing the total power consumed by the operating P4 devices.

B. Related Work

Since the introduction of NFV technology, many works were introduced to address the resource allocation and SFC embedding problems. More about these works can be found in these two surveys: [7] and [8]. For example, VNF-OP [9] presents a comprehensive problem formulation for the placement and mapping of SFC requests. The authors used commodity servers to host a variety of NFs that were instantiated based on the flow requirements. However, this paper does not consider function accelerators or P4 programmability as needed in our considered scenario.

While most solutions in the literature, such as VNF-OP [9], deal with NF placement on commodity servers, VNF-AAPC [10] considers also accelerators in the substrate when dealing with the placement problem. In the formulation of VNF-AAPC, the accelerator and server are considered as a single node. If a compatible accelerator is available, the serveraccelerator node uses fewer resources (i.e., cores) to host an NF. Although VNF-AAPC recognizes accelerators in the problem formulation, it does not consider P4 programmability and the fact that different accelerators may execute different functions, which requires proper mapping between the requirements of the NF workload and the availability of acceleration functions on the hosting devices.

In its problem formulation, P4NFV [11] takes P4 programmability into account. The authors of this paper are interested in determining the best NF placement between a SmartNIC and its hosting server. However, they do not consider the placement problem at the level of a full network which may be made up of heterogeneous types of P4 programmable devices. Smartchain [12] is another work that targets finding the optimal placement of an SFC between the SmartNIC and the CPU of a device, similar to [11], but without utilizing the P4 programmability.

Flightplan [13] enables the placement of P4 programs on a network of devices by utilizing coarse segmentation of the P4 programs. The model presented in Flightplan divides a P4 program into several smaller P4 programs that can run on different P4 targets. It is stated that this would improve the overall network performance and resource utilization. Although Flightplan takes into consideration the performance of segmented P4 programs, the applicability and usefulness of this performance awareness are hindered because of the coarse granularity in segmenting P4 programs compared to the approach followed in our work, where we consider the performance awareness at the level of atomic P4 operations.

LightNF [14] is a framework for NF offloading that considers its own set of NF primitives that are common to many implementations. While LightNF considers PISA switches as its offloading targets, the primitives are similar but distinct from the atomic operations introduced by the P4 language. Based on these identified NF primitives, called LightNF Primitives, the framework performs a code- as well as a static- and dynamic analysis on the SFC to be deployed to determine its structure, dependencies as well resource and performance demands. Subsequently, the optimizer maps the functions to either servers or switches aiming for either low latency or high throughput. As the set of primitives differs from P4’s the framework cannot be directly applied to the problem addressed in this work.

The authors of [15] introduce a decomposition and optimization framework for network functions based on P4. The approach does not consider SFC but rather individual NFs that the framework is meant to decompose into so-called µNFs. These µNFs are meant to be either deployed on containerized software switches running on servers, if they cannot be accelerated, or on P4 switch pipelines as a P4 program composed out of multiple µNFs. Compared to this work, P4 capabilities are treated in rather an abstract fashion where it is assumed that
any P4 device supporting the necessary functions and externs can be used to run a µNF. The heterogeneity of real device performance is not considered.

Lemur [16] is a cross-platform NF placement framework that aims to meet the defined service-level objectives for a given SFC. Code for individual functions is auto-generated for the chosen target platform. This means that Lemur requires a chain specification that results in the required functionality rather than mapping pre-existing NFs onto a substrate. This makes this approach very different from this work, where we consider existing NFs written in P4 and based on their functional and QoS requirements map them to an appropriate device for execution.

Finally, authors of [17] propose µP4 framework that enables composing multiple NFs to a single P4 program to be placed into any device. This work is complementary to our work on the implementation side. After the orchestrator decides on the optimal placement of NFs, the µP4 framework can be used to merge different NFs to be placed into a single P4 device.

III. PA-P4SFC-E Problem Formulation

In this section, we model the relevant parameters of the P4-enhanced NFV environment and then we describe the formulation of the studied PA-P4SFC-E problem. In Subsection III-A, we describe the modeling of the infrastructure of the environment and the capabilities of the constituting P4 packet processors. On the other end, the description of the workload in terms of NFs and SFCs requirements is illustrated in Subsection III-B. Finally, in Subsection III-C, we illustrate the formulation of the optimization problem, where decision variables, performance awareness feature, objective function, and constraints are described.

A. Infrastructure Variables and P4 Devices’ Characteristics

The target-independence feature of P4 allows the same P4 program to run on different types of P4 devices if a compatible P4 architecture is supported on these devices. The P4 architecture defines the programmable blocks in a P4 device, as well as the supported externs. These extern functions are additional methods supported by the P4 device that can be called from within a P4 program via a given API. For hardware P4 packet processors, externs can be used to access built-in acceleration functions such as an encryption function, while for software P4 packet processors, externs can be used to access software implementations of the referenced external function.

We define $D$, $D^P$, and $D^N$ to be the sets of all devices, all P4 programmable devices, and all non-programmable switches in a cloud environment, respectively ($D = D^P \cup D^N$). The set $D^P$ may include multiple instances of devices belonging to the same type of processing platforms such as FPGA-based P4 devices. In this case, these instances have the same device capabilities. The two sets $\mathcal{A}$ and $\mathcal{E}$ are defined to include all the supported P4 architectures, and the available P4 extern functions in a cloud environment, respectively.

Each P4 device $d \in D^P$ has the following characteristics:

- $\delta_{d}^{BP}$ stands for the base processing delay of P4 device $d$, which includes the delay of the non-programmable blocks in the device.
- $\delta_{d}^{e}$ stands for the processing delay of a P4 construct $e$ on P4 device $d$.
- $\delta_{d}^{f}$ stands for the processing delay of running a P4 extern function $e \in \mathcal{E}$ on P4 device $d$.
- $\omega_d$ stands for the processing resources of P4 device $d$ whose definition depends on the type of the processing platform. For example, while the processing resources in the case of ASIC devices are expressed in terms of the available Look-up-tables, these resources are expressed in terms of the number of available stages in the case of ASIC devices. To have a common definition, we quantify the processing resources of a P4 device as the maximum number of P4 constructs that can run simultaneously on that device.
- $\tau_d$ stands for the memory space in terms of the total number of rules that can be stored in a P4 device $d$.
- $P_d$ stands for the expected power consumption by P4 device $d$ when it is active. The power consumption of the device is assumed to be constant if the device is in use, and zero if it is not in use.
- $Arch_d^a$ is a Boolean parameter equal to 1 if P4 device $d$ supports architecture $a \in \mathcal{A}$, and equal to 0 otherwise.
- $Ext_d^e$ is a Boolean parameter equal to 1 if P4 device $d$ supports extern $e \in \mathcal{E}$, and equal to 0 otherwise.
- $T_d$ stands for the maximum supported throughput by P4 device $d$ on each of its interfaces.
- $T_d^e$ denotes the maximum supported throughput when running extern $e \in \mathcal{E}$ on P4 device $d$.

Each non-programmable switch $d \in D^N$ is characterized by a constant forwarding delay denoted by $k_d$. When these programmable P4 devices and the non-programmable switches are connected with physical links, they create a network. Let $\mathcal{X}$ denote the set of all physical links in the network and $T(d_i, d_j)$ denote the capacity of physical link $(d_i, d_j) \in \mathcal{X}$ for $d_i, d_j \in D$. Table I summarizes the description of all the symbols used for modeling the cloud infrastructure variables and P4 devices’ characteristics defined in this subsection.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Infrastructure Symbols Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D^P$</td>
<td>Set of P4 programmable devices</td>
</tr>
<tr>
<td>$D^N$</td>
<td>Set of non-programmable devices</td>
</tr>
<tr>
<td>$D$</td>
<td>Set of all devices $D = D^P \cup D^N$</td>
</tr>
<tr>
<td>$\mathcal{A}$</td>
<td>Set of possible P4 architectures</td>
</tr>
<tr>
<td>$\mathcal{E}$</td>
<td>Set of possible P4 extern functions</td>
</tr>
<tr>
<td>$\delta_{d}^{BP}$</td>
<td>Base processing delay on a P4 device $d \in D^P$</td>
</tr>
<tr>
<td>$\delta_{d}^{e}$</td>
<td>Processing delay of executing P4 construct $e$ on device $d \in D^P$</td>
</tr>
<tr>
<td>$\omega_d$</td>
<td>Available processing resources in device $d \in D^P$</td>
</tr>
<tr>
<td>$P_d$</td>
<td>Power consumption of P4 device $d \in D^P$ when active</td>
</tr>
<tr>
<td>$Arch_d^a$</td>
<td>Boolean parameter equal to 1 if device $d \in D^P$ supports architecture $a \in \mathcal{A}$</td>
</tr>
<tr>
<td>$Ext_d^e$</td>
<td>Boolean parameter equal to 1 if device $d \in D^P$ supports extern $e \in \mathcal{E}$</td>
</tr>
<tr>
<td>$T_d$</td>
<td>Supported throughput by P4 device $d \in D^P$ on each of its interfaces</td>
</tr>
<tr>
<td>$T_d^e$</td>
<td>Supported throughput when running extern $e \in \mathcal{E}$ on P4 device $d \in D^P$</td>
</tr>
<tr>
<td>$k_d$</td>
<td>Delay of non-programmable devices $d \in D^N$</td>
</tr>
</tbody>
</table>

TABLE I: Description of symbols used for modeling the cloud infrastructure variables and P4 devices’ characteristics.
B. Service Function Chains Requirements

The processing workload that needs to be supported by the cloud infrastructure is made up of SFCs. An SFC describes a processing path made up of more than one NF connected in a particular order. For example, it could be Firewall functionality followed by a Load Balancer, followed by L3Fwd. First, we describe and formulate the requirements of elementary NFs and then we illustrate the SFCs requirements formulation.

We denote \( \mathcal{F} \) to be the set that includes the NF instances workload that should be placed into the network. The set \( \mathcal{Y} \) includes all the possible types of NFs. For example, there could be more than one instance of Firewall NF in \( \mathcal{F} \), but all these NFs have the same type \( y \in \mathcal{Y} \). Note that some functionalities, such as the L2Fwd or L3Fwd, need to run on every used packet processor to guarantee proper packet forwarding between forwarding devices. The set of required functions is denoted by \( \mathcal{F}_{req} \). Set \( \mathcal{F}_{tot} = \mathcal{F} \cup \mathcal{F}_{req} \) includes all the NFs defined in a scenario.

The syntax of P4 language defines a basic set of P4 constructs or operations, denoted by set \( \mathcal{C} \). Any NF of type \( y \in \mathcal{Y} \) can be written as a combination of the P4 constructs contained in \( \mathcal{C} \). The set of P4 constructs that are needed to compose a NF instance of type \( y \) is denoted by \( \mathcal{C}_y \), where \( \mathcal{C}_y \subset \mathcal{C} \). The following variables summarize the requirements associated with any NF \( f \) of type \( y \):

1. \( \sigma_y^f \) represents the number of occurrences of each construct \( c \in \mathcal{C}_y \) needed to build a NF of type \( y \).
2. \( \omega_y^f \) represents the total number of P4 constructs required to describe a NF of type \( y \), i.e., \( \omega_y^f = \sum_{c \in \mathcal{C}_y} \sigma_y^f \). This parameter reflects the complexity of the NF, and consequently, the expected required processing resources by this NF.
3. \( Arch_y^f \) is a Boolean parameter equal to 1 if the NF of type \( y \) is compatible with P4 architecture \( a \in \mathcal{A} \), and equal to 0 otherwise.
4. \( Ext_y^f \) is a Boolean parameter equal to 1 if the NF of type \( y \) requires extern \( e \in \mathcal{E} \), and equal to 0 otherwise.
5. \( \tau_y^f \) represents the expected required number of rules that need to be stored when hosting NF instance \( f \in \mathcal{F} \).
6. \( QoS_f^s \) represents the expected throughput that needs to be processed by NF instance \( f \in \mathcal{F} \).

We define the set \( \mathcal{S} \) to contain all the SFCs that should be supported by the network. The set \( \mathcal{F}^s \) is defined to include all the NFs that build an SFC \( s \in \mathcal{S} \), while the set \( \mathcal{L}^s \) is defined to contain all the logical links between the NFs that constitute the SFC \( s \in \mathcal{S} \). The pair \( f_k, f_{k+1} \) stands for any two consecutive NFs in SFC \( s \in \mathcal{S} \).

A request to embed an SFC may be associated with certain QoS levels in terms of desired throughput or forwarding delay. The expected throughput associated with an SFC \( s \) is denoted by \( QoS_s^f \). The expected throughput to be processed by all NFs \( f \in \mathcal{F}^s \) that constitute an SFC \( s \) is equal to the required throughput to be processed by SFC \( s \), i.e., \( QoS_f^s = QoS_s^f \) for every \( f \in \mathcal{F}^s \). If an SFC does not have a QoS requirement in terms of throughput, a “best-effort” QoS level is assumed with throughput equal to a predefined small value. The QoS requirement for an SFC \( s \) in terms of the upper limit on the forwarding delay is denoted by \( QoS_s^d \). If an SFC does not have a maximum delay requirement, it is given a “best-effort” service with \( QoS_s^d \) set equal to a very large number.

Table II summarizes the description of all the symbols used for modeling NFs and SFCs workload and their requirements.

### Table II: Description of symbols used for modeling NFs and SFCs workload and their associated requirements.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Workload Symbols Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{F} )</td>
<td>Set of all NF instances to be placed</td>
</tr>
<tr>
<td>( \mathcal{F}_{req} )</td>
<td>Set of required NFs for forwarding</td>
</tr>
<tr>
<td>( \mathcal{F}_{tot} )</td>
<td>Set of all types of NFs</td>
</tr>
<tr>
<td>( \mathcal{Y} )</td>
<td>Set of all types of NF instances to be placed</td>
</tr>
<tr>
<td>( \epsilon_y^f )</td>
<td>Boolean parameter equal to 1 if NF of type ( y \in \mathcal{Y} )</td>
</tr>
<tr>
<td>( \omega_y^f )</td>
<td>Number of occurrences of each construct ( c \in \mathcal{C}_y ) needed to build an NF of type ( y )</td>
</tr>
<tr>
<td>( Arch_y^f )</td>
<td>Boolean parameter equal to 1 if NF instance ( f \in \mathcal{F} ) is compatible with P4 architecture ( a )</td>
</tr>
<tr>
<td>( Ext_y^f )</td>
<td>Boolean parameter equal to 1 if NF of type ( y ) is compatible with P4 architecture ( a )</td>
</tr>
<tr>
<td>( \mathcal{S} )</td>
<td>Set of all the Service Function Chains (SFCs)</td>
</tr>
<tr>
<td>( \mathcal{L}^s )</td>
<td>Set of all the logical links between NFs of ( s \in \mathcal{S} )</td>
</tr>
<tr>
<td>( \gamma_{k,d} )</td>
<td>Two consecutive NFs in SFC ( s \in \mathcal{S} )</td>
</tr>
<tr>
<td>( QoS_f^s )</td>
<td>Expected throughput that should be processed by SFC ( s )</td>
</tr>
<tr>
<td>( QoS_s^d )</td>
<td>QoS requirement for delay limit</td>
</tr>
</tbody>
</table>

C. Problem Formulation

The PA-P4SFC-E problem searches for the optimal embedding of a given workload of SFCs into a given P4 infrastructure while satisfying all the functional and QoS constraints associated with each SFC. This is enabled by the performance awareness feature which permits calculating a priori the forwarding latency of different candidate SFC embedding solutions. In the following, we first describe how the delay of different SFC embedding options is calculated and then we define the objective function and constraints relevant to this problem.

1) Decision Variables and SFC Delay Calculation: The decision variables in this problem specify where the constituting NFs and the logical links of an SFC are embedded. The first Boolean decision variable \( \alpha_{d_f}^s \) is equal to one if the \( k^{th} \) NF of SFC \( s \) is placed on P4 device \( d \). The second Boolean decision variable \( \gamma_{d_f}^s \) is for mapping the logical links of an SFC to the physical links of the network. It is equal to one if logical link \((f_k,f_{k+1}) \in \mathcal{L}^s \) utilizes the physical link \((d_i,d_j) \in \mathcal{X} \). Note that, unlike the physical links, the logical links of an SFC have direction. This is why the order of parameters in the \( \gamma \) subscripts and superscripts matters.

Using the previously provided formulation of the requirements of any NF of type \( y \in \mathcal{Y} \) and the capabilities of any P4 device \( d \in \mathcal{D}^p \), we can calculate the delay that will result from any NF placement option. This delay is calculated as the summation of the base processing delay of the hosting P4 device, denoted by \( \delta_d^p \), and the delay related to processing the programmable logic in the device, denoted by \( \Delta_d^y \). The latter delay component is expressed as follows:

\[
\Delta_d^y = \sum_{c \in C_y} \sigma_c^f \delta_c + \sum_{e \in \mathcal{E}[\text{Ext}f]} \delta_d^e, \quad (1)
\]
which is equal to the delay of processing the different P4 constructs that constitute the NF of type \( y \), and the delay for executing extern functions required by this NF.

For calculating the delay of different SFC embedding options, we take the following considerations:

- It is possible that the NFs of one SFC get placed on the same device or different devices across the network. Therefore, the delay calculation of an SFC should include the base delay, \( \delta^B_{d, P} \), of all devices used by this SFC. This base delay accounts for the delay to access a P4 device and process the non-P4 programmable blocks inside the device. In case consecutive NFs of the same SFC are placed on the same device, then the base delay is added only once for accessing the device.

- The average processing delay for running an arbitrary NF instance \( f \) of type \( y \) on device \( d \in D^P \) is denoted by \( \Delta^y_d \), and can be calculated according to Eq. (1).

- Some functionalities must run on each operating device in the network. For example, it is assumed for this problem that each operating P4 device should run L2Fwd functionality, if the device hosts a part of an SFC that split over multiple devices that need to forward packets within the network, to ensure proper forwarding between connected devices in the network. So the delay to execute this required L2Fwd functionality should be added to the total delay of an SFC if it traverses from one to another device. For the sake of generality, the set \( F_{req} \) is already defined to include all possible required functionalities. The delay to process the required NFs in the set \( F_{req} \) is calculated according to the following equation:

\[
\Delta^s_{req} = \sum_{f \in F_{req}, y \in Y} \psi^y_f \times \left( \sum_{c \in C_y} \sigma^c_{y,d} \delta^c_d + \sum_{e \in E[Ext^s_f] = 1} \delta^e_d \right) = \sum_{f \in F_{req}, y \in Y} \psi^y_f \times \Delta^y_d. \tag{2}
\]

- The propagation delay over the links is neglected because we are dealing with a scenario where the network is in a cloud data center with relatively short link delays.

- NFs constituting an SFC can be placed on separate devices. As a result, it is possible that packets need to be forwarded using devices in the network other than the hosting ones to complete their NF processing chain. While the forwarding delay of programmable devices \( D^P \) can be calculated as shown in Eq. 2, the forwarding delay over the non-programmable switches \( D^N \) is set to be a constant. All these forwarding delays are also added to the overall SFC delay calculation.

Taking these aspects into consideration, the total forwarding delay of a packet traversing an SFC \( s \) is calculated as shown in the following equation:

\[
\Delta^s = \sum_{d \in D^P} \left( \delta^B_{d, P} \alpha^f_{d, f} + \theta_{s, d, \sigma} \Delta^s_{req} + \sum_{f \in F^s} \sum_{y \in Y} \alpha^{|f|}_d \psi^y_f \Delta^y_d \right) + \sum_{d \in D^N} \theta_{s, d, k_d}. \tag{3}
\]

where \( \theta_{s, d, \sigma} \) and \( \theta_{s, d, k_d} \) are dependent variables that count the number of logical links corresponding to SFC \( s \) that enter and leave a P4 device \( d \in D \), respectively. These two dependent variables are calculated as follows:

\[
\theta_{s, d, \sigma} = \sum_{(f, f_{k+1}) \in L} \sum_{d_j \in D} \theta_{s, d, \sigma} \Delta^s_{d, d_j}, \tag{4}\]

\[
\theta_{s, d, k_d} = \sum_{(f, f_{k+1}) \in L} \sum_{d_j \in D} \theta_{s, d, \sigma} \Delta^s_{d, d_j}. \tag{5}\]

The delay components that contribute to the overall forwarding delay when traversing an SFC \( s \) shown in Eq. (3) are explained in the following:

- The term \( \sum_{d \in D^P} \delta^B_{d, P} \alpha^f_{d, f} \) counts for the base processing delay while accessing the first hosting P4 device, while the term \( \sum_{d \in D^P} \theta_{s, d, \sigma} \Delta^s_{d, d_j} \) counts for this base processing delay at all following P4 devices used for hosting SFC \( s \).

- The term \( \sum_{d \in D^P} \theta_{s, d, k_d} \) is equal to the sum of the processing delays of all required NFs when leaving a P4 device. In this scenario, the required NF is selected to be L2Fwd.

- The term \( \sum_{d \in D^P} \sum_{f \in F^s} \sum_{y \in Y} \alpha^{|f|}_d \psi^y_f \Delta^y_d \) is equal to the sum of the processing delays of the P4 programmable blocks of all the NFs in SFC \( s \) over all the used P4 devices.

As previously stated, SFCs can share NFs. If two SFCs have the same type of NF placed on one P4 device, then the placement of this function is done only once and the two SFCs can share the usage of this NF. This is necessary to save processing resources on the hosting P4 devices by avoiding duplicate placement of NFs of the same type. Also, note that L2Fwd is assumed as a required function on devices that host a part of an SFC that split over multiple devices, to ensure proper forwarding between connected devices in the network.

For illustration purposes, we elaborate on how the delay is calculated for an SFC distributed across three different P4 devices, as shown in Fig. 2. The SFC is made up of 4 NFs, the network is made up of CPU and ASIC-based P4 devices, and the racks are connected with a non-programmable switch. The placement of the NFs and logical links of this SFC is spread over three different P4 devices. The
delay calculation for this SFC is equal to the summation of the following components: (i) the base processing delay for accessing CPU1, the delay for processing NF1 on CPU1, and the delay to process L2Fwd function on CPU1 to forward packets to the second hosting device; (ii) the base processing delay to access ASIC1, the delay for processing NF2 and NF3 on ASIC1, and the delay to process L2Fwd function on ASIC1 to forward packets to the following hosting device; (iii) the constant forwarding delay \( k_d \) on the non-programmable switch; (iv) finally, the base processing delay to access ASIC2 and the delay for processing NF4 on ASIC2. There is no need to add the delay of L2Fwd at the end of the SFC because we assume that the last NF of any SFC should have L3Fwd functionality to guarantee proper routing of packets when leaving the network.

2) Objective Function: The objective of this problem is to minimize the operational costs when running the system. We select the total power consumption of active devices in the network, denoted by \( P \), as a representative of this cost, which is expressed as follows:

\[
P = \sum_{d \in D^P} P_d \times x_d,
\]

(6)

where the dependent variable \( x_d \), defined in Eq. (7), indicates whether the device \( d \in D^P \) is used in the embedding solution.

\[
x_d = \begin{cases} 
1 & \text{if } \sum_{f \in F_{tot}} \alpha_{d}^{f} \geq 1 \\
0 & \text{otherwise}
\end{cases}
\]

(7)

The model will favor using a combination of devices that consume less energy to reduce the total power consumption in the network. Eq. (8) depicts the objective function corresponding to the PA-P4SFC-E problem.

\[
\text{Minimize } (P).
\]

(8)

3) Constraints: The placement decision should recognize a set of constraints. First, it should be ensured that all the NFs \( f \in F \) that constitute the SFC workload have to be placed just once on any P4 device, as expressed in Eq. (9):

\[
\sum_{d \in D^P} \alpha_{d}^{f} = 1 \quad \forall f \in F.
\]

(9)

We make sure that a NF \( f \) of type \( y \) can be hosted on a device \( d \) only if the device has a compatible architecture and supports all the extern functions required by the NF of type \( y \). For this purpose, we introduce the variable \( \pi_{d}^{y} \) as a boolean variable equal to 1 if any NF instance of type \( y \) is placed on P4 device \( d \in D^P \). In Eq. (10), \( \pi_{d}^{y} \) is forced to be zero in case the architecture required by the NF and that supported by the P4 device do not match. Eq. (11) makes sure that \( \pi_{d}^{y} \) can be equal to 1 only if all the extern functions required by NF of type \( y \) are available on device \( d \). Note that \( N \) is a large constant bigger than the maximum number of externs required by any NF:

\[
\pi_{d}^{y} \leq \sum_{\alpha \in A} Arch_{d}^{\alpha} \times Arch_{y}^{\alpha} \forall y \in \mathcal{Y} \forall d \in D^P.
\]

(10)

\[
\sum_{c \in \mathcal{E}} Ext_{d}^{c} \leq \sum_{c \in \mathcal{E}} Ext_{d}^{c} + N (1 - \pi_{d}^{y}) \forall y \forall d \in D^P.
\]

(11)

Eq. (12) makes sure that the required processing resources of all NFs to be placed on any P4 device \( d \) never exceed the limited processing capacity of that device. The problem realizes that if two instances of NFs of the same type \( y \) are placed on the same device \( d \), then it is enough to place this NF only once to save processing resources on the hosting device. For this reason, the constraint limits the utilization of processing resources on a P4 device based on the requirements of different types of NFs hosted on that device, assuming that NFs of the same type will not be placed more than once on the P4 device.

\[
\sum_{y \in \mathcal{Y}} \pi_{d}^{y} \omega_{d} \leq \omega_{d} \forall d \in D^P.
\]

(12)

The required number of rules by all NFs to be placed on any P4 device \( d \) should never exceed the total memory space available on that device.

\[
\sum_{f \in F_{tot}} \alpha_{d}^{f} \tau_{f} \leq \tau_{d} \forall d \in D^P.
\]

(13)

The following two constraints are added to ensure proper forwarding between connected devices in the network by placing the L2Fwd NF as a required function on devices that host a part of an SFC that split over multiple devices. Note that \( M \) is an arbitrarily big number that should be greater than the maximum number of logical links leaving any P4 device.

\[
\alpha_{d}^{f} \leq \sum_{s \in S} \theta_{s,d}^{out} \forall d \in D^P \forall f \in F_{req}.
\]

(14)

\[
\sum_{s \in S} \theta_{s,d}^{out} \leq M \alpha_{d}^{f} \forall d \in D^P \forall f \in F_{req}.
\]

(15)

Eq. (16) ensures that the cumulative throughput of NFs that require offloading some functionalities to an extern function on a device never exceeds the limited throughput of that extern.

\[
\sum_{f \in F_{tot}} \sum_{y \in \mathcal{Y}} \alpha_{d}^{f} \times \psi_{d}^{y} \times Ext_{d}^{y} \times QoS_{f}^{T} \leq T_{d} \forall e \in E \forall d \in D^P.
\]

(16)

Moreover, we assume that each programmable device in the network has one ingress and one egress port to serve as an ingress or egress node for the traffic corresponding to any SFC. Hence, the cumulative traffic rate of all SFCs entering and leaving each programmable device on the ingress and egress ports, respectively should never exceed the capacity of these ports. This is achieved by the following two constraints:

\[
\sum_{s \in S} QoS_{s}^{T} \times \alpha_{d}^{f} \leq T_{d} \forall d \in D^P,
\]

(17)

\[
\sum_{s \in S} QoS_{s}^{T} \times \alpha_{d}^{f}_{end} \leq T_{d} \forall d \in D^P,
\]

(18)

where \( \alpha_{d}^{f}_{end} \) and \( \alpha_{d}^{f}_{end} \) respectively indicate whether the first (ingress) NF of SFC \( s \) and the last (egress) NF of \( s \) were placed on device \( d \).
TABLE III: Description of variables and objectives used to formulate the PA-P4SFC-E problem.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description for PA-P4SFC-E problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision Variables</td>
<td></td>
</tr>
<tr>
<td>$\alpha_d^f$</td>
<td>Boolean variable equals 1 if the $k$th NF of SFC $s$ is placed on P4 device $d \in D^e$</td>
</tr>
<tr>
<td>$\gamma_{f^s_k, f^s_{k+1}}$</td>
<td>Boolean variable equals 1 if logical link $(f^s_k, f^s_{k+1})$ uses the physical link $(d_i, d_j)$</td>
</tr>
<tr>
<td>Variables</td>
<td></td>
</tr>
<tr>
<td>$x_{d_i}$</td>
<td>Boolean variable equals 1 if any NF instance is placed on P4 device $d \in D^e$</td>
</tr>
<tr>
<td>$y_d^f$</td>
<td>Boolean variable equals 1 if any NF instance of type $y$ is placed on P4 device $d \in D^e$</td>
</tr>
<tr>
<td>$v^s_k$</td>
<td>Integer variable equals the number of logical links in SFC $s$</td>
</tr>
<tr>
<td>$\Delta_f$</td>
<td>Delay needed to process programmable logic (P4 blocks and externs) when running NF of type $y$ on P4 device $d$</td>
</tr>
<tr>
<td>$\Delta^*$</td>
<td>Average forwarding delay when traversing SFC $s$</td>
</tr>
<tr>
<td>Objective</td>
<td></td>
</tr>
<tr>
<td>$^P$</td>
<td>Total power consumed by active devices in the network</td>
</tr>
</tbody>
</table>

To make sure that all the logical links of all SFCs are mapped to physical links while respecting flow conservation, the following constraint is needed:

$$\sum_{d_j \in D \mid (d_i, d_j) \in X} (\gamma_{f^s_k, f^s_{k+1}} - \gamma_{f^s_{k-1}, f^s_k}) = (\alpha_{d_i}^f - \alpha_{d_j}^f) \quad \forall s \in S \quad \forall (f^s_k, f^s_{k+1}) \in L^s \quad \forall d_i, d_j \in D. \quad (19)$$

This constraint ensures that for any logical link between 2 consecutive NFs $(f^s_k, f^s_{k+1})$ of an SFC $s$, and for each device $d_i$, if the NF $f^s_k$ of SFC $s$ is not placed on $d_i$ as its predecessor $f^s_{k-1}$, then there should be a neighboring device $d_j$ such that the logical link between $f^s_k$ and $f^s_{k+1}$ uses its physical link with $d_j$. This equation also ensures that the sum of all incoming and outgoing flows of logical links for each device adds up to 0, i.e., the net flow is equal to 0 on all devices.

The following constraint is needed to ensure that a logical link cannot be mapped to the same physical link in both directions.

$$\gamma_{f^s_k, f^s_{k+1}} + \gamma_{f^s_{k-1}, f^s_k} \leq 1 \quad \forall s \in S \quad \forall (f^s_k, f^s_{k+1}) \in L^s \quad \forall d_i, d_j \in D \mid (d_i, d_j) \in X. \quad (20)$$

The capacity of physical links between devices should never be exceeded. This is ensured with the following constraint:

$$\sum_{s \in S} \sum_{(f^s_k, f^s_{k+1}) \in L^s} QoS_s^T \times \gamma_{f^s_k, f^s_{k+1}} \leq T(d_i, d_j) \quad \forall (d_i, d_j) \in X. \quad (21)$$

Finally, to support the QoS requirements in terms of forwarding delay associated with SFC embedding requests, the following constraint should be satisfied for all SFCs:

$$\Delta^* \leq QoS_s^D \quad \forall s \in S. \quad (22)$$

If an SFC does not have a delay QoS requirement (i.e., best-effort service), then $QoS_s^D$ is set to a very large number.

Table III summarizes the description of all the symbols used for the formulation of the PA-P4SFC-E problem.

IV. Greedy Algorithm for PA-P4SFC-E Problem

The SFC embedding problem is known to be an NP-hard problem. Therefore, it is important to consider a greedy approach that helps in solving the problem in a short time when the network is large even if this comes at the cost of reduced optimality. In this direction, we propose and implement a greedy algorithm as illustrated in Fig. 3.

The algorithm restricts the placement of an SFC to take place only on a single device, i.e., no SFC splitting on more than one device is allowed. This assumption is taken based on inspecting the optimal embedding solution selected by the ILP solver, wherein we found that the optimal solution rarely results in splitting the placement of SFCs into more than one device because of the performance penalty (additional base processing delay) incurred when accessing new devices. This assumption has a big impact on simplifying the solution because it saves searching for the optimal logical link placement. In this case, the link capacity throughput constraint is discarded and only the throughput constraints on the externs, ingress port, and egress port are applicable. However, we need to highlight that this assumption makes the greedy algorithm unable to find a feasible solution in cases when no single device can satisfy the processing requirements of the whole SFC, or in the case when the functional requirements of the distinct NFs of the SFC could not be met by a single device. In these cases, the operator can fall back to the optimal ILP-based solution described earlier.

When a new SFC request arrives, the algorithm searches for the best placement solution that reduces power consumption in the system while satisfying all the applied constraints. It is assumed that the execution time of the algorithm to embed incoming requests is faster than the inter-arrival time of these requests. This execution time is evaluated and shown to be in the milliseconds range, as reported later in Fig. 8b, which makes this assumption more reasonable. In the cases, when this assumption does not hold, the new request has to be buffered until the previous one is served.

In the ILP problem, this solution is found by the Branch and Bound Algorithm. In our proposed algorithm, the solution is found by applying the following steps as shown in Fig. 3:

1) **Excluding Device Types:** The algorithm checks first the constraints to exclude incompatible device types, such as CPU, NPU, FPGA, and ASIC, for hosting the given SFC. Then, it excludes all device instances belonging to a certain type in case this type does not satisfy all the constraints. These device type-related constraints include the P4 architecture compatibility, external requirements, and delay QoS requirement. Note that the last constraint depends on SFC delay calculation, which is the same for a given device type.

2) **Searching Reduced Space:** After shortlisting the candidate hosting P4 devices, the algorithm searches the reduced solution space by visiting an ordered list of devices, which is created to guide the search, instead of a random search, to reduce the execution time of the algorithm. The devices in the list are ordered in an ascending order based on a newly defined ratio equal to $\frac{\text{norm(Rate)}}{\text{norm(Power)}}$. This metric represents the power efficiency of a device per available resources.
resources. Thus, if the power consumption is smaller, or the device supports higher throughput capacity and has more processing resources, this ratio will be smaller and thus the device is more favored. This way, the SFC embedding solution is guided to minimize the power consumption in the system while recognizing the resources available on the devices and their throughput capacities. Note that this ratio is calculated for each device type after normalizing the constituting metrics (power, rate, and processing resources) between 0 and 1. This normalization is done by subtracting the minimum value recorded for that metric across different device types and dividing by the range of the values (i.e., maximum value - minimum value of the normalized metric among the device types). In the considered evaluation scenario, the device types are ordered according to the proposed ratio as follows: CPU, NPU, ASIC, FPGA, meaning that the placement algorithm will first favor checking the CPU devices, which have the lowest power consumption, as long as the other constraints, including delay QoS constraint, are satisfied. On the contrary, FPGA devices, with the highest power consumption and relatively fewer processing resources, will be checked as a last resort.

3) Checking Constraints: The next step is to check if the selected P4 device instance satisfies the remaining constraints which are applied for each device instance. These constraints include the throughput capacity of the devices as well as the processing/compute resources capacity. If all constraints are satisfied, then the embedding is successfully applied, otherwise, the next device in the ordered list from the previous step is examined.

4) Update Network Status & Excluding Device Instances: In this last step, the network status is updated by subtracting the resources consumed by the current SFC request from the hosting device. The ordered list is updated by excluding the device instances whose remaining processing resources or throughput capacity is smaller than preset thresholds, without changing the order of the devices. These preset thresholds are input parameters to the algorithm, whose values depend on the granularity of the SFC’s requests in terms of processing resources and throughput. In other words, these thresholds are set equal to the expected smallest throughput required in the system and the expected smallest SFC size (in terms of the total number of constituting P4 constructs) to be embedded in the system.

Table IV: Surveyed parameters of different used Network functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>L2Fwd</th>
<th>L3Fwd</th>
<th>FW</th>
<th>VDecap</th>
<th>LB</th>
<th>Tunneling</th>
<th>NAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Compat.</td>
<td>V1M,</td>
<td>V1M,</td>
<td>V1M,</td>
<td>V1M,</td>
<td>V1M,</td>
<td>V1M,</td>
<td>V1M,</td>
</tr>
<tr>
<td></td>
<td>SUME</td>
<td>SUME</td>
<td>SUME</td>
<td>SUME</td>
<td>SUME</td>
<td>SUME</td>
<td>SUME</td>
</tr>
<tr>
<td>Externs</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>PISP</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>Tc Univ.</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>Rule hdr.</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Modesty.</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Copy hdr.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Remove hdr.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Add hdr.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Add Table.</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

V. Evaluation

In this section, we design and evaluate different experiments to study the impact of different influential factors on the optimal solution of the PA-P4SFC-E problem, as well as to evaluate the proposed greedy solution. First, we elaborate on the selected values for populating the model’s parameters.

A. Surveying NFs’ and P4 Devices’ Parameters

To evaluate the proposed model, we select a scenario with a realistic set of NFs and P4 devices whose parameters are surveyed from the literature.

1) Surveyed NFs: We consider seven different NFs with different complexities: (i) Layer2 Forwarding (L2Fwd), (ii) Layer3 Forwarding (L3Fwd), (iii) Firewall (FW), (iv) VXLAN Decapsulation (VDecap), (v) Load Balancer (LB), (vi) Tunneling, (vii) Network Address Translation (NAT). The PA-P4SFC-E problem targets embedding SFCs made up of these NFs. The L2Fwd is selected to be in the set of functions required to be in every functioning device to ensure proper forwarding of packets within the network. All NFs support the three most common P4 architectures: V1M, SUME, and PISA. Only the LB NF requires the use of an external hashing function. Using the methodology described in [5], each NF is decomposed to its atomic P4 constructs. For example, L2Fwd NF necessitates parsing and modifying a single header (Ethernet), whereas L3Fwd necessitates the same operations for both Ethernet and IPv4 headers. Note that in the delay calculation of NFs, we subtract one from the number of parsed headers and one from the number of added tables when calculating the delay related to the P4 program to account for the presence of these two constructs in the base P4 program whose delay is included in the base processing delay component for each device.

Table IV summarizes the various parameters and requirements corresponding to all the considered NFs in this evaluation.

2) Surveyed P4 Devices: For this evaluation, we select four P4 programmable packet processors that belong to different types of processing platforms: CPU, NPU, FPGA, and ASIC. The different criteria related to the different selected types of P4 devices are summarized in Table V. The supported P4 architecture of each device is based on its datasheet. The throughput and the per P4 construct latency is selected based on our previous work [4], [5] and other literature [18]. Note that the delay values used in this paper are collected when devices are running at line rate, and accordingly, it is assumed that no queuing is taking place in the system’s devices. The
TABLE V: Surveyed parameters corresponding to P4 devices of different types.

<table>
<thead>
<tr>
<th>Param.</th>
<th>Device</th>
<th>CPU</th>
<th>NPU</th>
<th>FPGA</th>
<th>ASIC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P4 Architecture</td>
<td>V1M</td>
<td>V1M</td>
<td>SUME</td>
<td>PSA</td>
</tr>
<tr>
<td>Throughput per link (Gbps)</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Power Consumption (Watts)</td>
<td>50</td>
<td>65</td>
<td>184</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Max. Constructs</td>
<td>1000</td>
<td>500</td>
<td>250</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Extern (SipHash-2-4)</td>
<td>Present</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Throughput (Gbps)</td>
<td>3.3</td>
<td>7.6</td>
<td>4.2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Latency (µs)</td>
<td>90</td>
<td>40</td>
<td>0.5</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>Per P4 Construct Latency (µs)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base Processing Delay</td>
<td>45.9</td>
<td>7.45</td>
<td>3.54</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Parse Header</td>
<td>≈ 0</td>
<td>0.11</td>
<td>0.17</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Modify Header</td>
<td>0</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Copy Header</td>
<td>0</td>
<td>0.28</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Remove Header</td>
<td>-0.29</td>
<td>1.43</td>
<td>-0.02</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Add Header</td>
<td>0.4</td>
<td>1.59</td>
<td>0.23</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Add Table</td>
<td>0.08</td>
<td>0.37</td>
<td>0.13</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

performance related to the extern “SipHash-2-4” function is derived from [19] for packet size equal to 64 Bytes since the Load Balancer function requires calculating the hash function of some headers only. The support of this extern by a P4 device and its performance level (throughput and delay) determine whether the device can host an SFC requiring the extern with some required delay and throughput QoS levels. The maximum number of supported constructs is based on the comparative analysis provided in [18] and [20]. Since no work in the literature gives a quantitative evaluation regarding the maximum number of supported constructs on different devices, we assume some reasonable numbers that follow the order of devices in terms of the available processing resources.

Moreover, because no evaluation in the literature quantifies the variation of latency on ASIC devices as a function of P4 constructs and P4 externs, it is assumed that ASIC-based devices are powerful enough to run any P4 program with no latency variation and to execute hashing functions with higher performance than other device types. Finally, the power consumption of each device is surveyed from the literature and the datasheets of the devices. [21] states that the peak power consumption of the NetFPGA-SUME device can reach as high as 184 Watts (W). According to [22], the thermal design power, which reflects the theoretical maximum power consumption, of 4 cores CPU-based processor can reach up to 105 W. So we take the peak power consumption for running CPU-based P4 packet processors to be approximately equal to half this value since 2 cores are the typical minimum requirement for running P4-based software switches. Based on measurements reported in [23], the overall power consumption of a P4 ASIC-based switch, when measured at the plug, is approximately equal to 100 W. Finally, we assume that the power consumption of NPU-based P4 devices equals the power consumption of other NPU-based devices, whose peak power consumption is reported to be equal to 65 W according to its data sheet [24].

The web chart in Fig. 4 depicts a visual representation of the comparative advantages of the different devices in terms of different metrics [25]. The chart is based on the values presented in Table V after normalizing these values between zero and one based on the maximum value for each metric.

B. Evaluation Scenarios

We design three experiments to evaluate the formulated problem, whose design objective is described in the following:

1) The first experiment is meant to analyze the impact of the SFC workload characteristics (length of SFCs and required delay QoS) on the optimal solution.
2) The second experiment analyzes the impact of the infrastructure characteristics (degree of adoption of programmable ASICs) on the optimal solution.
3) The third experiment considers realistic scaled-up scenarios, wherein the greedy solution is also evaluated.

In all the experiments, the PA-P4SFC-E problem is formulated and implemented as an ILP problem and solved using the commonly used Gurobi solver [26].

The SFCs used in these experiments are chains made up of the NFs defined in Table IV. The workload to be placed is made up of a repeated set of SFCs, which are assumed to be distinct even though they belong to the same type because otherwise, NF sharing will take place reducing the overall workload to be placed into the network.

Note that the first two experiments are evaluated using small networks, where it is assumed that there are no functions running on the network before the placement optimization, and if any subset of the SFCs to be placed on the network could not be placed, the entire placement of the SFC workload is considered infeasible. On the other hand, in the third experiment, we consider a runtime scenario, where the model is required to embed SFCs into a bigger network already populated with other running SFCs. In other words, the problem takes into consideration the network state when it handles SFC embedding requests at runtime.

1) Experiment 1- Examining the Impact of SFC Characteristics: The goal of this experiment is to examine the impact of SFC characteristics on the embedding solution. For this purpose, we vary the length of the SFCs to be made up of 2, 4, and 6 connected NFs. Moreover, we consider 3 delay QoS levels for any SFC: 10 µs, 100 µs, and Best-Effort (no delay QoS requirement). To ensure a fair comparison when
examining the impact of SFC length on the performance of the system, we compare the three cases with different SFC lengths based on the total number of NFs to be embedded. The required throughput to be handled in the three cases is scaled properly such that the total throughput encountered by each of the three cases is the same, and the only difference is in the chaining complexity between the NFs based on the length of the SFCs.

The SFCs to be embedded are all made up of the NFs described in Table IV, and they all include LB NF that require SipHash-2-4 extern functionality. The evaluation is conducted assuming an infrastructure made up of different types of P4 devices as shown in Fig. 5a.

The results corresponding to this experiment are shown in Fig. 5. The plots in this figure show the value of the objective function in terms of the total power consumption in the system in Watts as a function of the total number of NF instances embedded into the system for different delay QoS cases. The different subplots 5b, 5c, 5d correspond to the cases where the length of the SFCs is set equal to 2, 4, and 6, respectively. Note that we plot on the x-axis the total number of NF instances constituting these SFCs to have a normalized way of presenting the processing workload of the different cases. In this evaluation, we gradually increase the number of SFCs to be embedded up to a point where no feasible solution could be found, i.e., the processing capacity of the network is reached.

For all SFC length cases, we can observe that when the delay QoS is more stringent, the total number of NFs (or SFCs) that are successfully embedded decreases. For example, looking at the case when the SFC length is equal to 6 in Fig. 5d, we can observe that the total number of NFs successfully embedded is equal to 78 NFs (=13 SFCs) when delay QoS is equal to 10 µs, which is less than the case when delay QoS is set to 100 µs where 126 NFs (=21 SFCs) could be embedded, which is less than the Best-Effort delay QoS case where 162 NFs (=27 SFCs) could be embedded. The reason for this is that when the required delay QoS is more stringent, the P4 devices with low performance cannot fulfill these requirements and thereby get shortlisted. For example, we can see from Table V that the base processing delay to access the CPU-based devices is approximately equal to 46 µs, accordingly, this device is automatically excluded from being an option to host SFCs with 10 µs delay requirement.

We can also observe from these figures that for a given NF workload, the total power consumption in the system increases when the delay QoS requirement is more stringent.

For example, looking at Fig. 5d, when 60 NFs (=10 SFCs) are to be embedded, the power consumption yielding from the optimal solution is equal to 280 W when delay QoS is equal to 100 µs, which is less than 568 W power consumption needed for hosting this workload when the delay QoS is equal to 10 µs. The reason for this is that the more performant P4 devices needed to suffice the embedding of SFCs with more stringent delay QoS requirements consume more power as can be inspected from Table V.

When comparing the three subplots in Fig. 5, we can observe that the impact of the length of SFCs on the optimal placement solution is minimal. Recalling that the incoming traffic rate per NF workload for all SFC lengths is normalized, we can conclude that the complexity of the SFCs in terms of the number of NFs chained does not affect the optimal orchestration solution and thus the system’s performance. Note that when the NF workload is high, we can observe a minor difference in the total number of NFs that can be placed into the network between the different SFC length cases. This is due to the difference in the granularity of the SFCs to be placed in terms of the number of constituting NFs and the throughput requirements. In other words, just before the saturation of the processing capacity of the P4 devices, it is possible to fit a few short SFCs with low throughput requirements into the remaining processing capacity of the system before saturation. This is not possible for longer SFCs with higher throughput requirements. Note that in most of the cases in this experiment, the capacity of the devices is reached because of the extern throughput constraint, i.e., the throughput capacity for running external functions.

2) Experiment 2- Examining the Impact of Programmable ASICs Adoption: The purpose of the second experiment is to study the impact of replacing traditional ToR switches with P4 programmable ASICs. These P4 ASIC devices can
execute NFs beside the typical forwarding. The experimental scenario is shown in Fig. 6, where the network is made up of CPU-based processors. The ToR switches can be replaced with P4 programmable ASIC switches according to the ratio “P4ASIC_Ratio”, denoted by \( r \), where a value of zero means that no P4 ASIC switches are used, while a value of one means that all ToR switches are replaced with P4 ASIC-based devices. Five networks are emulated where \( r \) is varied from zero to 1.

SFCs of length 4 with 1 Gbps required throughput QoS are selected as the workload to be embedded in this experiment, where the constituting NFs are based on those defined in Table IV such that all the chains include LB NF that require extern function. The delay QoS for the SFC workload is set to Best-Effort to make sure that CPU devices can suffice the delay requirements and thus can be used as potential hosts for the SFCs.

Fig. 7 shows the results corresponding to the optimal solution in experiment 2 in terms of the system’s power consumption as a function of an increasing SFC workload. Different plots correspond to different cases where the degree of adoption of P4 ASICs is varied between zero and one. The results corresponding to the different cases when P4 ASICs are used (i.e., \( r > 0 \)) resemble a similar pattern where the curves overlap up to a certain workload before they increase sharply. In the overlapping stage, the model makes use of the available P4 ASIC-based devices until they are fully occupied before falling back to the CPU-based devices. The reason for this is that the ASIC-based devices can handle higher throughput (\( x_1 \) normal traffic and \( x_6 \) traffic requiring extern function) compared to CPU-based devices as can be inspected from Table V. This makes the power efficiency of these ASIC devices higher compared to CPU-based servers since one ASIC device, even though consumes more power, can handle a larger workload than the power consumption of alternative CPU-based devices.

After the overlapping stage, the slopes of the curves corresponding to the cases when the ratio of P4 ASIC adoption is equal to 0.25, 0.5, 0.75, and 1 suddenly change after handling around 30, 60, 90, and 120 SFCs, respectively. At these points, the P4 ASIC devices are fully occupied, and the only other available type of devices, i.e., CPU-based, is used to handle the remaining SFC workload. It is observed that the slope of these curves after the sudden change is equal to that in the case when \( r = 0 \) where only CPU-based devices are available to host the SFC workload. When the throughput constraint on the CPU-based devices is reached, no more SFC embedding requests can be supported. Note that it is expected that the same patterns will get repeated when evaluating bigger networks with more racks and devices per rack.

In general, we can observe that using more programmable ASICs as ToR switches increases the embedding capacity of the system. Moreover, for the same SFC workload, using more programmable ASIC devices reduces the operational cost of the system presented as the power consumption. The selection of ASIC devices takes place, even though the power consumption of a single programmable ASIC device is higher than the power consumption of alternative CPU-based devices because a single ASIC device has a higher throughput capacity for accommodating a bigger number of SFCs, especially when extern function for LB NF is needed. However, it should be recalled that the performance gains when adopting ASIC devices come with a higher CAPEX cost as the programmable ASIC devices may be more expensive than traditional ToR switches.

3) Experiment 3- Evaluating the Greedy Solution: The purpose of this experiment is to evaluate the effectiveness of the proposed greedy solution in Section IV compared to the original ILP problem’s solution. The trade-off between the execution time and the optimality gap is analyzed. For this evaluation, we use a realistic SFC workload taken from the literature. We always assume that L3Fwd NF is available at the end of the SFCs to assure proper packet routing at the end of the processing chain. The adopted SFCs along with their delay and throughput QoS requirements are listed in the following: (1) Firewall → NAT → L3Fwd [27] with 10 \( \mu \)s delay QoS and 2 Gbps throughput QoS; (2) Load Balancer → L3Fwd [28] with 100 \( \mu \)s delay QoS and 1 Gbps throughput QoS; (3) Firewall → Load Balancer → L3Fwd [29] with Best-Effort delay QoS and 3 Gbps throughput QoS.

In this experiment, we consider 4 different topologies of increasing size to investigate the impact of scalability on the execution time of the algorithm. The first topology is made up of 4 racks, each with 40 CPUs and 40 IPUs (i.e., 40 FPGAs or 40 NPUs). The CPUs and IPUs per rack are connected with programmable ASIC ToR switches.

The other three evaluated networks are the same but with an increased number of racks: 10, 20, and 30 racks.
The results of this experiment are shown in Fig. 8. The power consumption in the system, when made up of 4 racks, in Watts, for both optimal and greedy solutions as a function of an increasing total number of NFs to be placed, is shown in Fig. 8a. Note that we show on the x-axis the total number of NFs instead of the total number of SFCs because the considered SFCs to be embedded in this experiment have different lengths. The embedding in this experiment is done at runtime, where the previous placements and network status are preserved when a new SFC embedding request is served. For this reason, we can see that the cumulative difference in the total power consumption in the system, compared to the optimal case, increases when the number of SFCs to be embedded increases as the per SFC embedding optimality gap accumulates. The total number of NFs that could be embedded in the system is around 1480 when the greedy solution is applied, approximately 26.6 % less than that when the ILP solution is applied, where the total number of NFs that could be embedded is equal to 2018. At the maximum number of NFs that could be embedded using the greedy solution, the optimality gap in terms of power consumption reaches around 8000 W.

Fig. 8b shows, in a logarithmic scale, the average execution time (in ms) needed for embedding an SFC request when the greedy and optimal solutions are applied as the number of racks in the network substrate increases. The standard deviation of the different recorded execution times is recorded and plotted on top of the averages. While the execution time to solve the ILP problem increases largely when the network size increases, our proposed greedy solves scaled-up scenarios faster. The optimal solution requires 435, 4137, 28500, and 64000 ms to be reached when the number of racks is equal to 4, 10, 20, and 30 racks, respectively, while the greedy solution only requires 5, 30, 35, and 53 ms to solve the problem in these cases. When looking at execution time results, it is clear that such a greedy solution is important, especially when considering large networks where tens of seconds are needed to find an embedding solution at runtime using the ILP formulation.

VI. CONCLUSION

In this work, we formulate and evaluate an optimization problem that targets finding the optimal embedding of SFCs into P4-enhanced NFV environments at runtime. The problem searches for the optimal placement and routing of SFCs into P4 programmable substrate, while fulfilling the functional and QoS requirements of the different SFC embedding requests. Moreover, the problem constraints the solution space to satisfy the requirements of the NF workload, while recognizing the distinct capabilities of the different candidates hosting P4 devices. Finally, a greedy solution is designed and implemented to solve the formulated problem faster at runtime. A detailed evaluation of the problem is conducted, where the model’s parameters are populated based on surveyed literature works. The impact of the different system parameters such as the length of SFCs and the degree of adoption of programmable ASICs is evaluated. Moreover, the evaluation of the greedy solution revealed the effectiveness of the solution when handling scaled-up scenarios in terms of the execution time, on the cost of reduced optimality.

Developing detailed models related to the power consumption of different P4 devices as a function of their activity and the running processing load is an interesting future research direction, especially when these models are utilized for optimizing the orchestration of P4-based cloud environments as done in this work. Moreover, this work can be extended by conducting evaluations related to utilizing the formulated optimization problem to orchestrate the placement of NF workloads in a real P4-based NFV environment. Finally, further enhancing the design of the greedy solution to achieve better performance is an interesting exercise to explore.

VII. ACKNOWLEDGMENT

This work has been partly funded by the European Commission through the H2020 project Hexa-X (Grant Agreement no. 101015956).

REFERENCES


VIII. BIOGRAPHIES

Hasanin Harkous is a System Architecture Research Engineer at Nokia working on 6G-related topics. He did his Ph.D. at Nokia Bell Labs in collaboration with the Technical University of Munich (TUM) in the Chair of Communication Networks. He received his Master’s Degree in Communications Engineering at TUM in 2018. His research focuses on topics related to programmable data planes, performance evaluation and modeling, and hardware acceleration in telco cloud technologies.

Bassel Aboul Hosn was born in 1998. He received his M.S. degree in communications engineering with distinction in 2022 from the Technical University of Munich (TUM), Germany. In the summer of 2018, he was a research intern at the Institute of Solid State Physics at the Technical University of Berlin (TUB). In 2021-2022, during his master’s studies, he worked on the topic of performance-aware P4 SFC embedding in the Chair of Communication Networks at TUM in collaboration with Nokia Bell Labs.

Mu He received his Master’s Degree in Communication Engineering from the Technical University of Munich (TUM) in 2015. He then became a research associate at the Chair of Communication Networks of Prof. Wolfgang Kellerer and received his Doctoral Degree in 2020. His research interests covered network planning in the context of software-defined networking, network optimization problems, P4 data plane, and runtime reconfiguration with programmable data plane devices.

Michael Jarschel received his Ph.D. degree from the University of Wuerzburg in 2014. Subsequently, he worked as a (Senior) Research Engineer in the area of network softwarization at Nokia (Bell Labs) in Munich until 2021. Currently, he is working as a professor of communication networks and cybersecurity at Technische Hochschule Ingolstadt. His main research interests are in the ongoing transformation of telecommunication systems towards software and the cloud, network security & early 6G concepts.

Rastin Pries is a research project manager at Nokia. He received his master and Ph.D. degree in computer science from the University of Wuerzburg, Germany in 2004 and 2010. His current research interests are in applying edge computing to cellular networks as well as localization and mapping approaches for real-time digital twinning.
Wolfgang Kellerer (M’96, SM’11) is a Full Professor with the Technical University of Munich (TUM), heading the Chair of Communication Networks at the Department of Electrical and Computer Engineering. Before, he was for over ten years with NTT DOCOMO’s European Research Laboratories. He currently serves as an associate editor for IEEE Transactions on Network and Service Management and as the area editor for Network Virtualization for IEEE Communications Surveys and Tutorials.