This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2021.3101366, IEEE Open Journal of the Communications Society

Network Function Decomposition and Offloading on Heterogeneous Networks with Programmable Data Planes

Daniele Moro, *Student Member, IEEE* Giacomo Verticale, *Member, IEEE*, Antonio Capone, *Fellow, IEEE*

Programmable network hardware is emerging as a viable option for offloading and thus accelerating network functions. However, the heterogeneous resources available in the network calls for a disaggregated deployment approach. Programmable switches, programmable Network Interface Cards (NICs), and in-network compute nodes exposes different peculiar resources and capabilities that can be maximally exploited only if the network functions are decomposed into multiple smaller network functions.

This work presents a framework for the automatic deployment of disaggregated and decomposed network functions. The framework comprises an orchestrator capable of deploying the decomposed network functions on programmable network hardware and software switches running in containers. The orchestrator exploits an optimization model for choosing the best decomposition according to the traffic demands, the network topology, and other constraints. An improved seamless deployment model and heuristic also accounts for the necessity of rerouting traffic when hardware nodes need to be re-programmed. Furthermore, the framework provides a tool to combine multiple functions into a single P4 program via a template pipeline that can be deployed to a programmable switch.

Numerical results highlight the advantages of offloading decomposed network functions to programmable network hardware. Furthermore, we show how the seamless deployment model and heuristic have negligible effects on the allocation time and accepted traffic requests while guaranteeing the rerouting of traffic when switches are put in maintenance mode.

***Index Terms*—Network Function Virtualization, Software Defined Networking, Programmable Networks**

I. Introduction

5G and Beyond 5G technologies are pushing unprecedented changes to the network architectures. Critical applications are deployed at the edges of the network to meet latency requirements and to keep sensitive data in trusted locations. Moreover, network technology is embracing a programmability approach that enables efficient traffic manipulation and computing tasks directly in hardware pipelines traditionally used only for switching, while software approaches such as eBPF/XDP are becoming increasingly important.

These options open new opportunities for deploying data plane Virtual Network Functions (VNFs) that can scale to multi-gigabit per second speeds. When VNFs need to process high traffic, they can be decomposed onto multiple, smaller functions, called μVNFs, and deployed on multiple physical nodes. In this way, the μVNFs can exploit the heterogeneous resources present in the network on programmable switches, programmable Network Interface Cards (NICs), and in-network compute nodes. When compared to deploying monolithic VNFs on general-purpose hardware, this approach results in lower jitter and lower risk of packet reordering. There are, however, some limitations in what operations can be done on programmable network hardware. For example, the Intel Barefoot Tofino provides a few pipeline stages to implement the packet processing functionalities, minimal QoS functionalities, and few megabytes of register memory. Conversely, programmable NICs can provide bigger memories to implement custom and complex hierarchical scheduling policies, but at the cost of slower packet processing. In order to exploit the flexibility and the advantages of having multiple hardware and software architectures in the network, an orchestrator must identify how to distribute each VNF on the available nodes, jointly finding the best decomposition and, for each μVNF, the implementation that can most efficiently exploit the available programmable hardware and software resources.

In addition, deploying new functions to programmable hardware requires putting the node in maintenance mode, resulting in prolonged packet losses and, consequently, traffic choking and rerouting. In order to avoid network downtimes, it is, therefore, necessary to schedule hardware reprogramming and proactively reroute all traffic flows during the maintenance along pre-provisioned paths providing the same VNFs. If such network paths are not available, then the orchestrator will not use that node for new VNFs.

This paper provides the following contributions.

- An orchestrator capable of deploying the μVNFs written in the P4 language [1] onto a network comprising both programmable hardware and P4 software switches running in containers. The orchestrator uses the features provided by the ONOS network controller for discovering the network topology and distributing the routing information to the switches.
- A toolset to combine multiple μVNFs into a single P4 program that can be instantiated on a programmable network device. This toolset is integrated into the orchestrator.
- A full deployment optimization algorithm for selecting a specific decomposition of each VNF. The deployment model also chooses the placement of each μVNF onto the network nodes, simultaneously determining whether it should be combined with other μVNFs and deployed on P4 hardware, or it should be deployed as a software μVNF. The optimization goal is to exploit as much as possible the heterogeneous resources of the programmable network devices.
- An improved and seamless version of the optimization al-

This work is funded by EU Grant no. 101016577, project AI-SPRINT.

A preliminary version of this paper appears in: D. Moro, G. Verticale, and A. Capone, "A framework for network function decomposition and deployment," in Proceedings of 16th International Conference on the Design of Reliable Communication Networks (DRCN) 2020

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2021.3101366, IEEE Open Journal of the Communications Society

gorithm which also tries to place the hardware μVNFs in such a way that traffic flows can be temporarily rerouted if the switch needs reprogramming. In case this is not possible, the switch is tagged as non-reprogrammable.
- A heuristic algorithm, faster version of the above seamless deployment algorithm.
- Numerical results obtained with simulations showing how deploying programmable network hardware and decomposed VNFs can be beneficial with respect to pure monolithic software solutions and how the improved optimization using the heuristic algorithm has negligible impact on the allocation times while guaranteeing better usage of the programmable resources.

The paper is structured as follows. Section II describes the state of the art in Network Function Virtualization (NFV) decomposition and placement and reports on similar efforts in the literature. Section III describes our assumptions and the scenario we considered. In Section IV we describe the components of the proposed framework at a high level and how they are linked together. Section V discusses the set of tools we developed to perform the deployment of the solution and the combination of different μVNFs into the same programmable switch. Section VI describes the full deployment optimization model used to identify and route the best VNF decomposition. In Section VII the seamless deployment model is described and Section VIII reports our numerical evaluation of the optimization models. In Section IX and Section X the heuristic for the seamless deployment and the comparison with the optimization model are presented. Finally, Section XI provides concluding remarks.

II. Related Work

In the recent years, many researchers have analyzed offloading use cases of packet processing logic to programmable network hardware. Authors of [2] propose to optimize the load of Deep Packet Inspection (DPI) software by offloading the connection tracking logic on programmable switches based on the Open Packet Processor (OPP) model [3], [4] uses the same programming model to accelerate Linux iptables. Other works exploit the P4 Language [1] as the means to offload the packet processing to programmable hardware. In [5], the authors offload the load balancing logic with P4, [5] instead proposes to implement a key-value store on programmable switches. Other scenarios consider data center coordination services [6] or MapReduce acceleration [7] by exploiting programmable data planes. In this work, we generalize the offloading idea to the case of generic VNFs. We additionally propose optimization models and a heuristic to optimally allocate the required VNFs as software components running on general-purpose servers or as a P4 program running on programmable network hardware.

The problem of composing P4 programs by having network primitives that can be reused independently from the network device target has been addressed in [8] where the authors focus on the single P4 device. Flightplan [9] is a framework allowing the disaggregation of a monolithic P4 program into many subprograms executed across multiple heterogeneous devices. Our work, instead, focuses on the composition of already pre-disaggregated programs on multiple network devices and general-purpose compute servers. Gallium [10] and P^{2}GO [11] propose two different methods to exploit both P4 programmable hardware and general-purpose CPUs. Gallium uses C++ VNFs and synthesizes them in P4, while P^{2}GO exploits traffic traces to decide whether to allocate part of a VNF in programmable hardware or as a software counterpart. In our work, we use pre-defined P4-based μVNFs, letting the optimization model decide where to allocate the required VNF based on the available network resources. Other works like HyPer4 [12], P4Visor [13], and HyperVDP [14] propose techniques to merge P4 programs to be executed on a single device. Our work proposes to achieve the coexistence of multiple decomposed P4 μVNFs via a template P4 pipeline. A dispatcher then uses the SRv6 Segment Identifier to decide which sub-function to execute.

Many researchers have tackled the optimal deployment of VNFs. In [15], the authors were among the first to study the joint problem of optimal placement and routing of VNFs, while in [16] the authors consider the cost of multi-core VNFs implementation in their linear model. The authors of [17] consider the VNF decomposition into multiple different implementations in their model, but they do not consider the possibility of having programmable data planes as a deployment target. This paper proposes optimization models and a heuristic to consider the decomposed VNF deployment on a heterogeneous network composed of both programmable network hardware and general-purpose compute servers with different costs and constraints depending on the type of node.

III. System Model

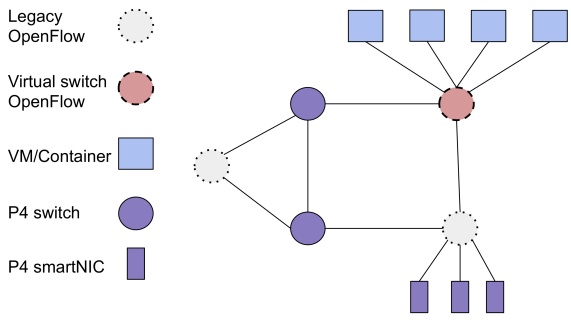

Fig. 1. Example of a heterogeneous network with programmable data planes. The nodes in the upper right corner represent a general-purpose compute server running in a data center, comprising a virtual switch and a set of VMs/container nodes.

In this work we consider a heterogeneous network, as shown in Figure 1. Multiple types of nodes are considered, in particular:

- P4-programmable hardware: in this category falls both programmable switches and programmable smartNICs. The nodes are characterized by their programmable pipeline. We consider both the width (i.e., number of pipeline stages that can be executed one after the other) and the depth (i.e., number of entries that can be installed

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2021.3101366, IEEE Open Journal of the Communications Society

in a table) of the programmable pipeline. We also consider the specific functions provided by the nodes (the so-called `extern` functions in the P4 Language) as a set of boolean flags characterizing the programmable devices.
- "Legacy" OpenFlow [18] switches: nodes that can be controlled via OpenFlow protocol. This set of nodes are used as dumb traffic forwarders.
- General-purpose compute servers: these nodes can dynamically spawn new P4 programmable virtual switches (e.g., in a container). We consider the virtual switches as devices with unlimited width and length of the programmable pipeline. Furthermore, we consider support for all the `extern` functions. Every general-purpose compute node is also equipped with a virtual OpenFlow switch in charge of forwarding the traffic among the different containers and the physical network interface connecting them to the rest of the network.

The network also includes an SDN controller responsible for configuring the network devices to route the traffic end-to-end. In this work, we choose the ONOS SDN controller [19]. ONOS is responsible for inserting the flow entries in the OpenFlow switches, and deploying and configuring the tables of the P4-programmable network devices.

IV. The Framework

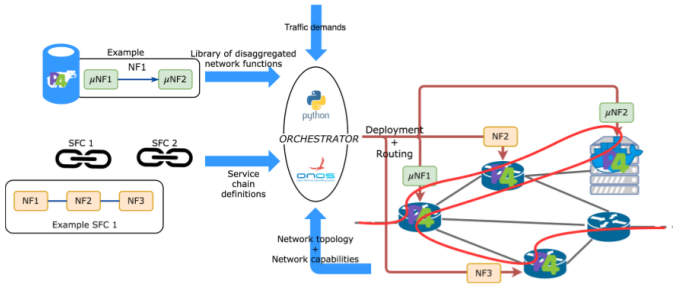

Fig. 2. The proposed framework and its components

The goal of the framework is the deployment of Network Functions (NFs) on an heterogeneous network with programmable data planes. The NF are combined to form a Service Function Chain (SFC) that is provided to the framework as a traffic demand. Figure 2 shows the components of the framework. For each NF, the catalog contains several decompositions into multiple μVNFs. Some of these could be additionally decomposed into smaller μVNFs as shown in Figure 3. Examples of μVNFs can be found in [8] and in [13]. Additionally, for each μVNF, the catalog may contain multiple implementations for different target hardware, characterized by different requirements in terms of width and depth of the pipeline and of the required `extern` functions. This paper assumes that the decompositions and implementations are stored in a catalog by means of P4 source files and are provided or pre-generated by the developer. It is worth noting that such decompositions could be generated on the fly from a single source file as illustrated in [9]. Using P4 as the reference language provides several advantages, such as multiple compilers targeting different devices. Currently, the reference P4C compiler [20] provides back-ends for deployment in the BMv2 software switch, as eBPF (extended Berkley Packet Filter) filters [21], and on DPDK (Data Plane Development Kit) Software Switch pipeline [22]. Furthermore, multiple switching chips vendors provide their own P4 compilers. For example, the Intel's P4Studio for the Intel Barefoot Tofino ASIC [23], the NetFPGA->P4 Toolchain exploits Xilinx's SDNet compiler [24] for the NetFPGA-SUME board, or the Netronome P4 Compiler for their Network Flow Processor [25].

The main component of the framework is the orchestrator, written in Python. This component receives as input the SFCs definition and traffic demands, the catalog of the NFs, and gathers the network topology information via ONOS REST APIs. The main goal of the orchestrator is choosing the optimal decomposition of traffic requests. Given the traffic demand for each SFC, the orchestrator formulates and solves an optimization model and chooses the best decomposition of the SFC, choosing from the available μVNFs in the catalog, the location of the required NFs, and the traffic routing on the network. We support two optimization models, one for the full deployment of network functions (Section VI) and another one for the seamless deployment, presented in Section VII. Furthermore, we also propose a heuristic algorithm, introduced in Section IX, to improve execution time with respect to the seamless deployment model.

The orchestrator, with the output from the optimization phase, combines the selected μVNFs that need to be deployed on the same device via a template P4 pipeline (described in Section V). It also compiles the generated P4 programs for the target device on the network selected by the optimization algorithm. Finally, the orchestrator deploys the selected μVNFs and the flow tables via ONOS.

While the framework is general enough to support any network function expressed in P4, it is best suited for the user-plane class of functions rather than its control-plane counterpart. The control plane network functions, e.g., the 5G 3GPP Session Management Function (SMF), usually manage low throughput of traffic and implement complex execution logic (i.e., authentication, authorization, session management). These functions are best suited to be implemented using a high-level language. The user plane functions, e.g., the 5G 3GPP User Plane Function (UPF), directly treat the user-generated traffic, implementing elementary operations such as filtering, tunnelling, header manipulation, or encryption, but dealing with very high throughput. These functions are the perfect fit for hardware-based packet processing and to be described in P4.

In the context of the ETSI NFV architecture [26], the proposed orchestrator fulfils multiple roles. The orchestrator realizes the *NFV Orchestrator* role by receiving SFC requests and orchestrating the required VNFs by decomposing them to exploit the available resources maximally. The orchestrator also manages the life cycle of the VNFs during the deployment phase, partially realizing the role of the *VNF Manager*.

V. Deployment

The orchestrator identifies the programmable network or general-purpose compute nodes where the μVNFs should be

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2021.3101366, IEEE Open Journal of the Communications Society

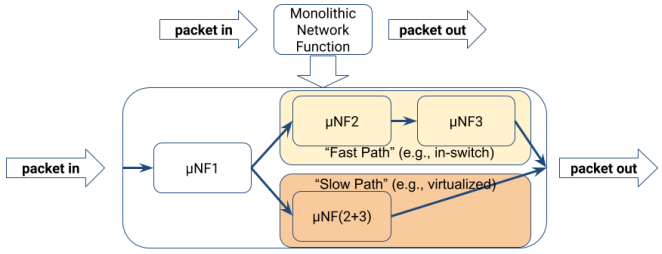

Fig. 3. Example of VNF decomposition. A single monolithic network function can be decomposed in several smaller network functions.

deployed with the optimization model. The SFC traffic is then steered through these waypoints with Segment Routing version 6 (SRv6) [27]. We use the SRv6 Network Programming Internet draft [28] as a reference. The SRv6 segments are a 128-bit IPv6 address called Segment Identifier (SID). The SIDs are used to identify both the node and function in the network. The destination of the packet is encoded in the 64 most significant bits, the 64 least significant bits, instead, identify the μVNF to be executed on the destination node. Figure 4 shows an example of the SRv6 header and the SIDs. The flow rules installed in the network devices and the SIDs carried in every packet are directly generated by the orchestrator.

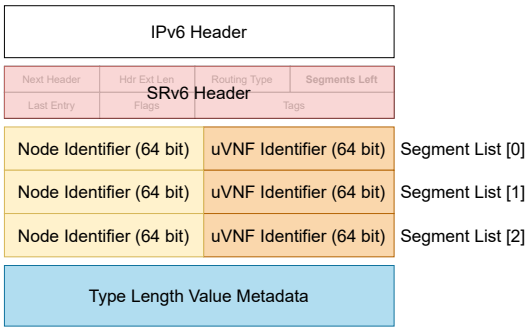

Fig. 4. The SRv6 header.

The generation of the P4 program is performed in the deployment phase. The generated P4 programs are then compiled and deployed on both the hardware and software P4-programmable nodes. The reference P4 architecture we consider is V1Model. This architecture, shown in Figure 5, is composed of four programmable components: the parser, the ingress pipeline, the egress pipeline and the de-parser. The traffic manager instead is currently not P4-programmable and for this reason not considered in our model.

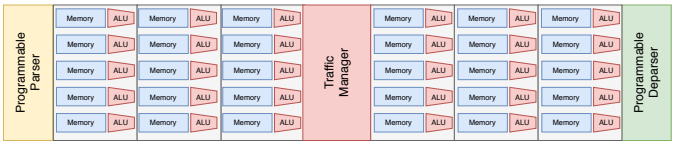

Fig. 5. V1Model Pipeline

In the proposed architecture, the parser and the de-parser components of the P4 pipeline are common to all the devices. The routing and the forwarding functionalities implemented in the ingress pipeline are the same for all the nodes. In this way, the developer of the μVNF does not have to re-implement these components in every new network function. Instead, we define a template pipeline used by the orchestrator to plug-in the required μVNF provided by the network functions catalog. The orchestrator composes the μVNFs on the template pipeline and creates the final P4 program that is then compiled and deployed on the network nodes.

The parser of the proposed template pipeline supports IPv6 packets with SRv6 headers. The ingress pipeline is composed of three stages. Figure 6 illustrates the high-level schema of the reference pipeline. The egress pipeline is omitted from the figure since it is currently unused.

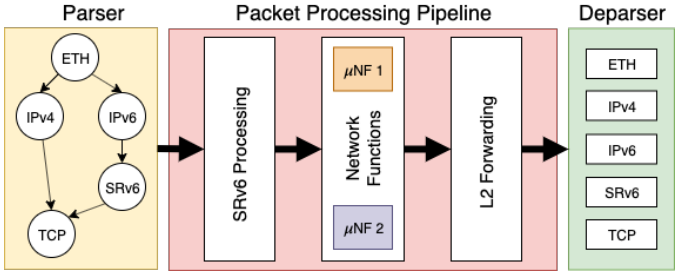

Fig. 6. Pipeline of the P4 programmable switch supporting μVNFs deployment. For the sake of simplicity, the traffic manager and egress pipeline are not shown.

The first stage of the template pipeline is in charge of the SRv6 processing. This stage first matches the IPv6 destination address in the `srv6_end` table. This table uses only the 64 most significant bits to identify if the SRv6 packet is destined to the current node. If the match is positive, the network function identifier is extracted from the 64 least significant bits. Then, the IPv6 destination address is overridden with the next SID extracted using the `Segments Left` header field. Finally, we have the `ipv6_next_hop` table in charge of selecting the destination of the current packet (i.e., setting the output port).

In the second stage, the template pipeline selects the μVNF to execute. Using the network function identifier extracted at the previous stage, a dispatcher triggers the corresponding μVNF. This function can then perform its packet processing functionality, and, besides that, it can also override the next hop (set in the previous stage). The SRv6 metadata shown in Figure 4 can be used to transfer state across functions deployed on different devices. The network functions can read and write these metadata because they are parsed when the packet is received, and deparsed before sending the packet out towards the next μVNF in the chain. This part of the P4 pipeline is dynamically generated by the orchestrator when combining the μVNFs selected in the optimization phase.

Finally, in the last stage, layer 2 packet forwarding operations are performed. If the current node is not the packet's destination (i.e., no positive match in the `srv6_end` table), the device forwards the packet towards the destination based on the L2 header information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2021.3101366, IEEE Open Journal of the Communications Society

The final step is the compilation of the generated P4 programs. The orchestrator is in charge of compiling each program for the specific hardware. In this paper, we do not consider the compilation time of the resulting P4 program a critical issue, because compilation can be performed in the background with no service interruption. However, there are some technologies for which compilation times can become significantly large as the number of VNFs grows. In those cases, the allocation model can be easily modified to take complexity into account and avoid selecting deployments that result in too many VNFs in the same programmable switch.

To deploy the resulted output on the software nodes, the orchestrator spawns a new container running a P4 virtual switch in the general-purpose compute node and connects it to the local OpenFlow virtual switch. Then, it deploys the compiler output on the new spawn virtual node. In the case of programmable hardware devices, the orchestrator directly installs the compiler output on the device. Finally, ONOS receives the routing information and installs the flow rules into all the network nodes (both OpenFlow and P4-programmable). Our framework implementation exploits BMv2 to emulate the hardware devices and the software counterparts running on the Docker containers. The combination of P4 programs has been implemented in Python and released as an open source contribution together with the P4 template pipeline we use for small scale testing [29].

In the following sections we present two deployment models. The *full deployment model* assumes that the orchestrator can reprogram a P4-programmable node with no or negligible downtime. In the *seamless deployment model* we acknowledge that reprogramming a node requires steering its traffic along another route and putting it in maintenance mode. Consequently, the latter model avoids choosing programmable nodes if an alternate route for the current traffic cannot be found.

VI. Full Deployment Optimization Model

The following mixed integer linear optimization, called full deployment model, is used to choose the optimal decomposition of a single SFC on the available network resources, together with the routing of traffic between μVNFs.

a) Sets:

U: Set of all the physical nodes (OpenFlow-only, with programmable hardware, with CPUs, and hosts)
$U = U_{OF} \cup U_{HW} \cup U_{SW} \cup U_{HOST}$

E: Set of all the physical links
$E = \{e_{u,v} | u, v \in U\}$

G: Physical topology
$G = (U, E)$

VNF: Set of all the possible VNFs and μVNFs of the SFC

DC: Set of all the possible decompositions

V_{dc}: Set of VNFs or μVNFs in each decomposition
$V_{dc} \subseteq VNF \quad \forall dc \in DC$

E_{dc}: Set of virtual links in each decomposition
$E_{dc} = \{e_{i,j} | i, j \in V_{dc}\} \quad \forall dc \in DC$

G^{dc}: Virtual graph
$G^{dc} = (V_{dc}, E_{dc})$

$SRC \in U_{HOST}$ Traffic source of the SFC

$DST \in U_{HOST}$ Traffic destination of the SFC

b) Parameters:

$S_{dc} \in VN_{dc} \quad dc \in DC$ Source of the decomposition

$D_{dc} \in VN_{dc} \quad dc \in DC$ Destination of the decomposition

tr Traffic request of the SFC

$BW_{e_{u,v}}$ Bandwidth available on the link (u, v)

$extern_i$ Set of P4 “extern" functions required by VNF i

ex_u Set of P4 “extern" functions supported by node u

$depth_i$ Depth of pipeline stages required by function i

$width_i$ Width of the pipeline stages required by function i

d_u Maximum pipeline depth supported by node u

w_u Maximum pipeline width supported by node u

CPU_u Number of CPUs in node u

C_{SW} Cost (per unit of traffic) of deploying a function on a software node

C_{HW} Cost of deploying a function on a hardware node

c) Variables: Indicator variable: it is 1 if dc is the chosen decomposition.

$$z_{dc} \in \{0, 1\} \quad \forall dc \in DC \tag{1}$$

Indicator variable: it is 1 if function i is deployed on node u.

$$x_{dc,i,u} \in \{0, 1\} \quad \forall dc \in DC, i \in V_{dc}, u \in U \tag{2}$$

Indicator variable: it is 1 if virtual link (i, j) is deployed on the physical link (u, v).

$$f_{dc,e_{i,j},e_{u,v}} \in \{0, 1\} \quad \forall dc \in DC, e_{i,j} \in E_{dc}, e_{u,v} \in E \tag{3}$$

d) Objective function: The objective function is composed of three terms to account for the use of the software, hardware nodes, and to account for the path cost. The cost associated to the software nodes is proportional to the traffic associated to the requested SFC. The cost associated to the hardware nodes instead is independent to the requested traffic. When using programmable hardware nodes, the traffic is always forwarded at line rate if the P4 program for that node can be compiled (i.e., there is enough resources available on the programmable pipeline). The cost associated to the path is needed to enforce that the selected path is the shortest one, by minimizing the cost associated to the path itself.

$$\min \quad \text{cost}_{SW} + \text{cost}_{HW} + \text{cost}_{PATH} \tag{4}$$

with

$$\text{cost}_{SW} = \sum_{\substack{dc \in DC \\ i \in V_{dc} \\ u \in U_{SW}}} x_{dc,i,u} tr \cdot C_{SW} \tag{5}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2021.3101366, IEEE Open Journal of the Communications Society

$$\text{cost}_{HW} = \sum_{\substack{dc \in DC \\ i \in V_{dc} \\ u \in U_{HW}}} x_{dc,i,u} C_{HW} \tag{6}$$

$$\text{cost}_{PATH} = \sum_{\substack{e_{i,j} \in E_{dc} \\ dc \in DC \\ e_{u,v} \in E}} f_{dc,e_{i,j},e_{u,v}} \tag{7}$$

e) Decomposition constraints: These constraints enforce that only one, of the many available, decomposition is selected.

$$\sum_{dc \in DC} z_{dc} = 1 \tag{8}$$

Enforce that only the µVNFs of the selected decomposition are chosen.

$$\sum_{u \in U} x_{dc,i,u} = z_{dc} \quad \forall dc \in DC, i \in V_{dc} \tag{9}$$

The following constrains, instead, enforce that the source and sink of the SFC traffic are the chosen one.

$$x_{dc,S_{dc},SRC} = z_{dc} \quad \forall dc \in DC \tag{10}$$

$$x_{dc,D_{dc},DST} = z_{dc} \quad \forall dc \in DC \tag{11}$$

$$\sum_{u \in U, u \neq SRC} x_{dc,S_{dc},u} = 0 \quad \forall dc \in DC \tag{12}$$

$$\sum_{u \in U, u \neq DST} x_{dc,D_{dc},u} = 0 \quad \forall dc \in DC \tag{13}$$

f) Constraints on node types: Only P4 programmable hardware devices and general-purpose compute nodes can allocate functions. Standard OpenFlow devices and hosts can not allocate µVNFs.

$$\sum_{\substack{dc \in DC \\ i \in V_{dc} \\ u \in U_H \cup U_{OF}}} x_{dc,i,u} = 0 \quad \forall dc \in DC \tag{14}$$

g) Capacity constraints: Enforce the availability of the required externs with respect to the externs required by the µVNFs.

$$\begin{aligned} x_{dc,i,u} \leq 1 \quad & \forall dc \in DC \\ & \forall (i,u) \in \{(i,u) | ex_u \subseteq extern_i, i \in V_{dc}, \\ & dc \in DC, u \in U\} \end{aligned} \tag{15}$$

Enforce the availability of pipeline depth and width on hardware and software nodes.

$$\sum_{\substack{dc \in DC \\ i \in V_{dc}}} depth_i \cdot x_{dc,i,u} \leq d_u \quad \forall u \in U_{hw} \cup U_{sw} \tag{16}$$

$$\sum_{\substack{dc \in DC \\ i \in V_{dc}}} width_i \cdot x_{dc,i,u} \leq w_u \quad \forall u \in U_{hw} \cup U_{sw} \tag{17}$$

For software nodes, we need also to enforce that enough CPU capacity is available. The CPU capacity required by every µVNF is proportional to the pipeline depth associated to that µVNF.

$$\sum_{\substack{dc \in DC \\ i \in V_{dc}}} depth_i \cdot tr \cdot x_{dc,i,u} \leq CPU_u \quad \forall u \in U_{sw} \tag{18}$$

h) Link to path mapping: This set of constraints are needed to map a logical link of the virtual graph into a physical path.

$$\sum_{e_{u,v} \in E} f_{dc,e_{i,j},e_{u,v}} - \sum_{e_{v,u} \in E} f_{dc,e_{i,j},e_{v,u}} = x_{dc,i,u} - x_{dc,j,u} \quad \forall dc \in DC, e_{i,j} \in E_{dc}, u \in U \tag{19}$$

i) Bandwidth constraint: Enforce that enough bandwidth is available in every selected physical link.

$$\sum_{\substack{dc \in DC \\ e_{i,j} \in E_{dc}}} tr \cdot f_{dc,e_{i,j},e_{u,v}} \leq BW_{e_{u,v}} \forall e_{u,v} \in E \tag{20}$$

VII. Seamless Deployment Optimization Model

The full deployment model described in Sec. VI allows multiple SFCs to share the same node to allocate different µVNFs. When considering an online allocation model, however, we need to consider that reprogramming an hardware node requires putting the the node in maintenance mode. As shown in [30], updating a P4 program running on a state-of-the-art programmable switching chip can take up to 50ms, that in the case of a 100Gbps link translated to more than 7 million packets lost per switch port. To alleviate this issue, we can temporarily move the µVNFs allocated on the hardware node by re-deploying it on unused network capacity. While the µVNF is moved, the corresponding SFC traffic is re-routed to the new node, and the orchestrator can deploy the new set of µVNFs. For this reason, we extend the full deployment model described in Section VI obtaining the seamless deployment model. This optimization model accounts for the possibility of maintenance mode in hardware nodes, achieved by embedding the same SFC two times on the same physical network. The primary deployment is used to allocate the SFC on the physical network. The shadow deployment, instead, is calculated to guarantee that the current µVNF can be moved to other network resource when required. The guarantee is "soft" in terms of available resources, we are not reserving any spare capacity on the network for eventually move the SFC. The guarantee is instead "hard" in the topological sense. We want to be sure that there is a disjoint hardware nodes path in order to re-deploy the current SFC when the maintenance mode is required.

a) New and Extended Sets:

V'_{dc}: Shadow set of VNFs or µVNFs in each decomposition
V'_{dc} is a duplicate of V_{dc}

E'_{dc}: Shadow set of links in the virtual graph
E'_{dc} is a duplicate of E_{dc}

$\widehat{G}^{dc}$: The extended virtual graph, including the shadow graph
$\widehat{G}^{dc} = (\widehat{V}_{dc}, \widehat{E}_{dc}) = (V_{dc}, E_{dc}) \cup (V'_{dc}, E'_{dc})$

b) Updated Variables: Indicator variable: it is 1 if function i is deployed on node u.

$$x_{dc,i,u} \in \{0,1\} \quad \forall dc \in DC, i \in \widehat{V}_{dc}, u \in U \tag{21}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2021.3101366, IEEE Open Journal of the Communications Society

Indicator variable: it is 1 if virtual link (i,j) is deployed on the physical link (u,v).

$$f_{dc,e_{i,j},e_{u,v}} \in \{0,1\} \quad \forall dc \in DC, e_{i,j} \in \widehat{E}_{dc}, e_{u,v} \in E \quad (22)$$

c) Decomposition constraints: Constraint (9) has been extended to account for the deployment of both the primary and shadow embedding.

$$\sum_{u \in U} x_{dc,i,u} = 2 \cdot z_{dc} \quad \forall dc \in DC, i \in V_{dc} \quad (23)$$

d) Constraints on node types: The constraint (14) need to be applied to the $\widehat{V}_{dc}$ set, to make sure that also in the "shadow" deployment only P4 programmable hardware nodes and software nodes can allocate functions.

$$\sum_{\substack{dc \in DC \\ i \in \widehat{V}_{dc} \\ u \in U_H \cup U_{OF}}} x_{dc,i,u} = 0 \quad \forall dc \in DC \quad (24)$$

e) Capacity constraints: This class of constraints needs to be enforced on the $\widehat{V}_{dc}$ set, to make sure that also in the "shadow" embedding there is sufficient capacity in terms of width and depth of pipeline, and the required externs for the shadow deployment are available in the hardware and software nodes.

$$x_{dc,i,u} \leq 1 \quad \forall dc \in DC \quad \forall (i,u) \in \{(i,u) | ex_u \subseteq extern_i, i \in \widehat{V}_{dc}, dc \in DC, u \in U\} \quad (25)$$

$$\sum_{\substack{dc \in DC \\ i \in \widehat{V}_{dc}}} depth_i \cdot x_{dc,i,u} \leq d_u \quad \forall u \in U_{hw} \cup U_{sw} \quad (26)$$

$$\sum_{\substack{dc \in DC \\ i \in \widehat{V}_{dc}}} width_i \cdot x_{dc,i,u} \leq w_u \quad \forall u \in U_{hw} \cup U_{sw} \quad (27)$$

$$\sum_{\substack{dc \in DC \\ i \in \widehat{V}_{dc}}} depth_i \cdot tr \cdot x_{dc,i,u} \leq CPU_u \quad \forall u \in U_{sw} \quad (28)$$

f) Link to path mapping: This constrain need to be extended to account for the $\widehat{E}_{dc}$ set. This allow to map both the logical "primary" and "shadow" links into a path in the physical topology.

$$\sum_{e_{u,v} \in E} f_{dc,e_{i,j},e_{u,v}} - \sum_{e_{v,u} \in E} f_{dc,e_{i,j},e_{v,u}} = x_{dc,i,u} - x_{dc,j,u} \quad \forall dc \in DC, e_{i,j} \in \widehat{E}_{dc}, u \in U \quad (29)$$

g) Bandwidth constraint: The bandwidth constraints are kept as in the standard model. The "shadow" path does not need to have any bandwidth constraints enforced on the selected path, because of the soft-guarantee on the "shadow" deployment.

h) Shadow path constraint: This new class of constraints is needed to make sure that hardware nodes used by "primary" ("shadow") path are not used by the "shadow" ("primary") path. These constrains make also sure node-disjointness on the hardware nodes, indeed we do not want any hardware nodes used for the "primary" ("shadow") deployment to be used on the opposite.

The following two constraints make sure that hardware nodes used for embedding the "primary" deployment are not used for the "shadow" path:

$$\sum_{\substack{e_{i,j} \in E'_{dc} \\ e_{u,v} \in E}} f_{dc,e_{i,j},e_{u,v}} + \sum_{\substack{e_{i,j} \in E'_{dc} \\ e_{v,u} \in E}} f_{dc,e_{i,j},e_{v,u}} \geq 0 \text{ if } x_{dc,y,u} = 0 \quad \forall y \in V_{dc}, dc \in DC, u \in U_{HW} \quad (30)$$

$$\sum_{\substack{e_{i,j} \in E'_{dc} \\ e_{u,v} \in E}} f_{dc,e_{i,j},e_{u,v}} + \sum_{\substack{e_{i,j} \in E'_{dc} \\ e_{v,u} \in E}} f_{dc,e_{i,j},e_{v,u}} = 0 \text{ if } x_{dc,y,u} = 1 \quad \forall y \in V_{dc}, dc \in DC, u \in U_{HW} \quad (31)$$

The following two constraints enforce the opposite, making sure that hardware nodes used for embedding the "shadow" deployment are not used for the "primary" path:

$$\sum_{\substack{e_{i,j} \in E_{dc} \\ e_{u,v} \in E}} f_{dc,e_{i,j},e_{u,v}} + \sum_{\substack{e_{i,j} \in E_{dc} \\ e_{v,u} \in E}} f_{dc,e_{i,j},e_{v,u}} \geq 0 \text{ if } x_{dc,y,u} = 0 \quad \forall y \in V'_{dc}, dc \in DC, u \in U_{HW} \quad (32)$$

$$\sum_{\substack{e_{i,j} \in E'_{dc} \\ e_{u,v} \in E}} f_{dc,e_{i,j},e_{u,v}} + \sum_{\substack{e_{i,j} \in E'_{dc} \\ e_{v,u} \in E}} f_{dc,e_{i,j},e_{v,u}} = 0 \text{ if } x_{dc,y,u} = 1 \quad \forall y \in V_{dc}, dc \in DC, u \in U_{HW} \quad (33)$$

The seamless deployment model, being more restrictive than the full deployment model can fail to allocate a traffic request. In this case we fall-back to the full deployment model with the constraint that the hardware nodes selected for deploying the current traffic request are marked as non-programmable and will not be used by any subsequent deployment until the current traffic demand is removed from the network.

VIII. Numerical Results of the Optimization Models

In this Section, we report on the performance of the full and seamless deployment models over a few relevant architectures. The results are based on simulations. We used the Gurobi solver for solving the optimization problem while generating multiple random instances. We first briefly present the results of the full deployment model over different topology configurations. We compare central cloud deployment and edge cloud deployment with and without P4 programmable devices. The extended results on the full deployment model are available in [31]. Finally, we compare the full deployment model and the seamless deployment model over the best

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2021.3101366, IEEE Open Journal of the Communications Society

topology configuration to understand how performance and time complexity results are affected when accounting for the traffic re-routing.

A. Full Deployment Model with Different Topology Configurations

For these simulations we used *BT-Europe* topology from the Internet Topology Zoo [32]. We considered four different topology scenarios, namely:

- ***DC***: all the SFC requests can be only deployed in a single large data center. This scenario simulates a remote cloud scenario.
- ***μDC***: the SFCs can be deployed on multiple smaller data centers, simulating an edge-cloud scenario. One-third of nodes have computational capabilities.
- ***P4-DC***: the same as *DC* but half of the other nodes are P4 programmable hardware.
- ***P4-μDC***: the same as *μDC*, with half of the other nodes that are P4 programmable.

In the two scenarios with programmable P4 hardware, we consider smartNICs and programmable switches as the two possible kinds of hardware. The smartNICs provide a pipeline depth of 20 and a pipeline width of 10. Switches, instead, have a pipeline depth of 100 and a pipeline width of 50. In the network we consider a number of P4 programmable switches that is double with respect to the number of smartNICs.

In each simulation, we generate 150 consecutive SFC requests. Each SFC is taken from a pool of 10 random SFCs. The traffic requested by every SFC is between 1 and 50, and the number of network functions is uniformly distributed between 1 and 10. The number of different decomposition of every VNF is distributed uniformly between 2 and 5, and the number of μVNFs composing every decomposition instance is uniformly generated between 1 and 5. The μVNFs are characterized with a pipeline occupancy specified in width and depth, generated randomly between 1 and 20, and 1 and 15, respectively. Simulation parameters are similar to the parameters used in [17], but considering that we use P4-based devices as target in our implementation.

Figure 7 shows how the network configurations that includes programmable hardware are able to accept more SFCs than the one with only data centers and thus general-purpose computing devices. Results in [31] also demonstrate that the total cost of the accepted SFCs grows linearly with the number of offered SFCs in the *DC* and *μDC* scenarios. Adding P4 programmable devices allows allocating a significant number of SFCs at a small cost, increasing the network's overall capacity.

The robustness of the various deployment scenarios has also been tested. We evaluate how network capacity changes when links are removed from the network itself. Figure 8 shows how the network configuration with μDC has the least capacity loss both with and without P4 programmable hardware. More information about the simulations and results with the full deployment model can be found in [31].

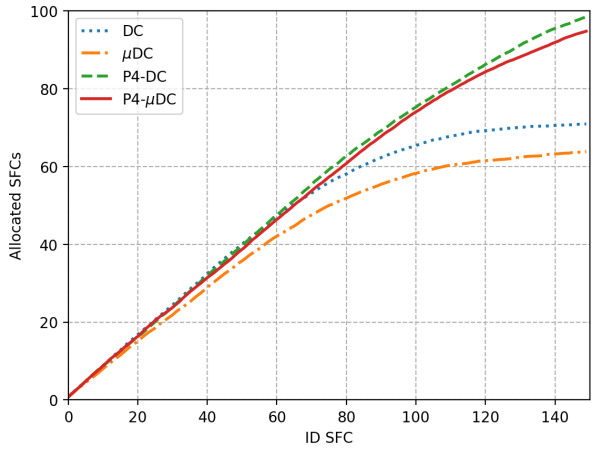

Fig. 7. Number of accepted SFCs vs number of offered SFCs for the four topology scenarios.

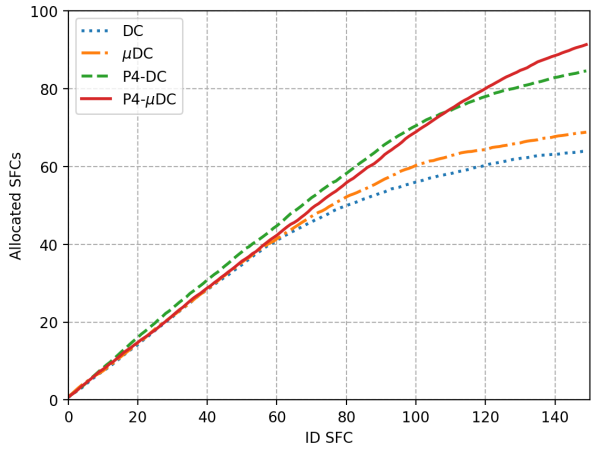

Fig. 8. Number of accepted SFCs vs number of offered SFCs for the four topology scenarios after the removal of 2 links.

B. Comparison Between Full and Seamless Deployment Models

We consider the BT-Europe topology, and we select the *P4-μDC* scenario, which yields the best performance. We run an online simulation with a birth-death process for the SFC requests. We assume the SFC requests arrive in a Poisson process, and the lifetime of each SFC request is exponentially distributed. We simulated the process with a fixed average lifetime of 1000 time units and with varying average arrival rates from 2 to 8 SFC requests per 100 time units, similarly to simulations done in [17]. The simulations are performed for 20000 time units.

The SFC requests are generated from a pool of 20 pre-generated SFCs. Each SFC requests an amount of traffic between 1 and 70 traffic units. The number of network functions within a single SFC is uniformly distributed between 2 and 4. Each VNF has a random number of possible implementations between 2 and 5. The number of μVNFs composing a VNF implementation is uniformly generated between 2 and 5. Each μVNF is characterized by a pipeline occupancy in depth and width; these parameters are generated at random between 1

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2021.3101366, IEEE Open Journal of the Communications Society

and 15, and 1 and 20, respectively. Each simulation is repeated 35 times.

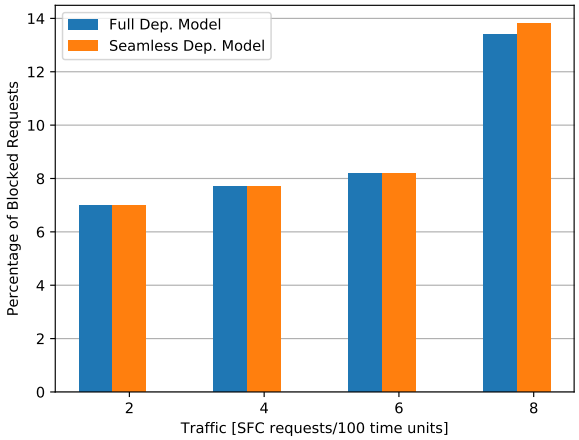

Fig. 9. BTEurope block rate with increasing offered traffic

Results in Figure 9 show how the seamless deployment model is able to provide a shadow deployment with little to no loss in terms of accepted SFC requests with respect to the simplified deployment model. The block rate increases up to 0.5% with 8 SFC arrivals every 100 time units, and no block rate differences with lower offered traffic is observed. This guarantees that an SFC can be re-routed while a new μVNFs is being allocated on a hardware node.

However, the seamless deployment model, being more constrained, requires a longer time to allocate a single SFC request. The time needed by the optimization model improves of more than 7 times, passing from around 0.10 s to allocate a single SFC to about 0.74 s.

IX. Heuristic for Seamless Deployment

The seamless deployment model, while providing a negligible loss of accepted SFC requests, causes a considerable increment of the time needed to allocate each SFC request, as described in Section VIII. For this reason, we develop a heuristics for the seamless deployment of the SFCs, illustrated in Algorithm 1. The heuristic is based on the full deployment model and works in 3 steps:

1) *Primary Deployment*: Exploiting the full deployment model illustrated in Sec. VI, the SFC request is embedded on the given network. In this way, we find the primary deployment for the current SFC;
2) *Reduced Graph*: From the physical network graph, the hardware nodes used for the primary deployment are removed, obtaining a reduced graph that is used in the following step;
3) *Shadow Deployment*: We re-embed the same SFC request on the reduced graph again with the full deployment model. From this embedding, we obtain the shadow deployment of the SFC request.
   - If the shadow deployment is unavailable because of a lack of resources on the reduced graph, the algorithm tags the hardware nodes used for the primary deployment as non-reprogrammable. In this way, no other μVNFs are going to be deployed on those nodes unless the current SFC request is removed from the network.

Figure 10 shows an example of how the heuristic can be applied for the deployment of an SFC.

Algorithm 1: Heuristic for seamless deployment

Input: $graph$ = Physical Topology
SFC = SFC Request
// Primary deployment
$primary$ = Full Dep Model$(graph, SFC)$;
// Remove resources used on primary
// deployment
$reduced\_graph$ = Reduce Graph$(graph, primary)$;
// Re-embed the same SFC on the
// reduced graph
$shadow$ = Full Dep Model$(reduced_graph, SFC)$;
if *shadow not available* **then**
  // If the Full Dep Model fails
  // in finding the shadow path,
  // lock nodes used in the primary
  // deployment
  $locked\_graph$ = Lock Nodes$(graph, primary)$;
  return $locked\_graph, primary$
else
  return $graph, primary$
end if

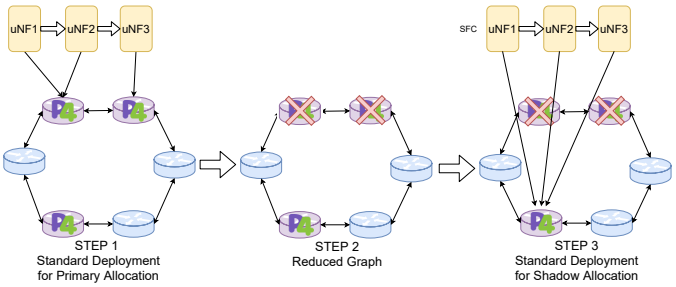

Fig. 10. The heuristic for SFCs deployment.

X. Numerical Results of Heuristic for the Seamless Deployment

In this section we present the results of the heuristic for the seamless deployment of SFCs compared to the full and seamless optimization models previously described.

A. Setup

We used three different simulation topologies, namely *BT-Europe*, *Abilene*, and *Interroute* from the Internet Topology Zoo [32]. We simulated the *P4-μDC scenario* as in Section VIII-B with similar parameters that we briefly recap. The birth-death process is run for 20000 time units. We simulate increasing average arrival rate between 2 and 8 SFC requests

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2021.3101366, IEEE Open Journal of the Communications Society

every 100 time units with an average lifetime of 1000 time units for the BT-Europe and Interroute topology and 500 time units for the Abilene topology. The SFC requests are generated from 20 pre-generated SFCs. Each SFC requests a traffic amount between 1 and 70 traffic units. The number of VNFs in a single SFC is uniformly distributed between 2 and 4. Each VNF has a number of different possible implementations between 2 and 5. The number of µVNF composing a VNF implementation is uniformly generated between 2 and 5. Each µVNF has a pipeline occupancy in depth and width between 1 and 15, and 1 and 20, respectively.

B. Results

Figure 11 shows the block rate with increasing offered traffic on the BT-Europe topology. As expected from the previous results, the optimization models and the heuristic algorithm behave similarly at low offered traffic. In this condition, the shadow path required during maintenance mode can always be found without rejecting any SFC. When the traffic increases, for example, starting from 6 SFC requests every 100 time units, the seamless deployment model and heuristic start to block more SFCs. In particular, as we already mentioned in Section VIII, the seamless deployment model blocks up to 0.5% more SFC than the full deployment model. The heuristic instead blocks up to 2.5% more SFC requests than the optimization model. The heuristic, however, provides a massive improvement on the optimization time required to allocate a single SFC request. As shown in Table I, the seamless deployment model leads to a rise of the execution time of 7 times with respect to the simplified deployment model. The heuristic, instead, increases the execution time of up to 1.7 times while still guaranteeing the shadow deployment needed for maintenance mode with a low loss in accepted SFC requests.

TABLE I
TIME REQUIRED FOR RUNNING THE OPTIMIZATION MODEL AND THE HEURISTIC TO ALLOCATE A NEW SFC REQUEST

Topology	Full Deployment Model	Seamless Deployment Model	Seamless Deployment Heuristic
Abilene	0.047 s	0.241 s	0.076 s
BT-Europe	0.103 s	0.743 s	0.174 s
Interroute	0.577 s	6.414 s	1.254 s

Figure 12 shows the results with the Abilene topology. This topology resembles a trap topology, which could create issues when looking for node disjoint paths. The optimization models and heuristic perform similarly to the BT-Europe topology, with a comparable increment of block rate and comparable optimization time growth from the simplified model as shown in Table I.

Finally, we test the optimization models and heuristic with a bigger topology. In particular with the Interroute topology. Also in this case, as can be seen in Table I, the heuristic provides a huge improvement in the execution time, allowing the correct deployment of SFC in a heterogeneous network even with a very big network deployments. Indeed, the deployment

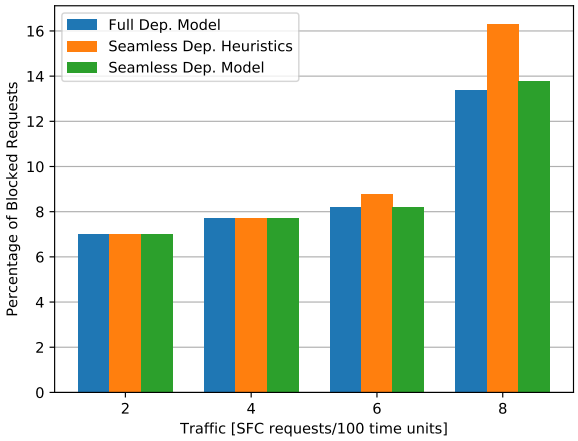

Fig. 11. BTEurope: block rate with increasing offered traffic

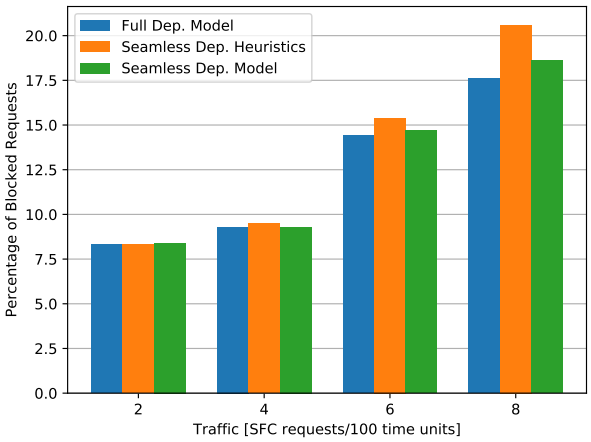

Fig. 12. Abilene: block rate with increasing offered traffic

of a single SFC request with the seamless deployment model can take up to 6.4 s, while with the heuristic takes around 1.25 s.

XI. CONCLUSION

This paper presents a framework for deploying disaggregated network functions, called µVNFs, on heterogeneous networks. The orchestrator exploits ONOS to deploy the µVNFs on both programmable network hardware and P4 programmable software switches running in containers. The framework combines multiple µVNFs into a single P4 program via a template pipeline and allocates them on programmable switches. We propose an optimization model to place the µVNFs on the available network resources accounting for possible re-routing when hardware nodes need to be re-programmed with a seamless deployment model. We also propose a faster heuristic algorithm for seamless µVNFs placement. The routing of the packets between µVNFs exploits SRv6 and follows the IETF SRv6 Network Programming draft.

Results show that VNFs disaggregation and programmable network hardware allow to allocate more VNFs with respect to pure software solutions. We also show that the seamless deployment model and heuristic have a negligible impact both on time and on the number of accepted SFC requests while guaranteeing the maintenance mode for network nodes during the re-deployment of μVNFs.

REFERENCES

[1] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2656877.2656890

[2] D. Sanvito, D. Moro, and A. Capone, "Towards traffic classification offloading to stateful sdn data planes," in *2017 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2017, pp. 1–4.

[3] G. Bianchi, M. Bonola, S. Pontarelli, D. Sanvito, A. Capone, and C. Cascone, "Open packet processor: a programmable architecture for wire speed platform-independent stateful in-network processing," *arXiv preprint arXiv:1605.01977*, 2016.

[4] L. Petrucci, M. Bonola, S. Pontarelli, G. Bianchi, and R. Bifulco, "Implementing iptables using a programmable stateful data plane abstraction," in *Proceedings of the Symposium on SDN Research*. ACM, 2017, pp. 193–194.

[5] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 15–28.

[6] X. Jin, X. Li, H. Zhang, N. Foster, J. Lee, R. Soulé, C. Kim, and I. Stoica, "Netchain: Scale-free sub-rtt coordination," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 35–49.

[7] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-network computation is a dumb idea whose time has come," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. ACM, 2017, pp. 150–156.

[8] H. Soni, M. Rifai, P. Kumar, R. Doenges, and N. Foster, "Composing dataplane programs with μp4," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 329–343. [Online]. Available: https://doi.org/10.1145/3387514.3405872

[9] N. Sultana, J. Sonchack, H. Giesen, I. Pedisich, Z. Han, N. Shyamkumar, S. Burad, A. DeHon, and B. T. Loo, "Flightplan: Dataplane disaggregation and placement for p4 programs," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/sultana

[10] K. Zhang, D. Zhuo, and A. Krishnamurthy, "Gallium: Automated software middlebox offloading to programmable switches," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 283–295. [Online]. Available: https://doi.org/10.1145/3387514.3405869

[11] P. Wintermeyer, M. Apostolaki, A. Dietmüller, and L. Vanbever, "P2GO: P4 Profile-Guided Optimizations," in *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, ser. HotNets '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 146–152. [Online]. Available: https://doi.org/10.1145/3422604.3425941

[12] D. Hancock and J. van der Merwe, "Hyper4: Using p4 to virtualize the programmable data plane," in *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 35–49. [Online]. Available: https://doi.org/10.1145/2999572.2999607

[13] P. Zheng, T. Benson, and C. Hu, "P4visor: Lightweight virtualization and composition primitives for building and testing modular programs," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 98–111. [Online]. Available: https://doi.org/10.1145/3281411.3281436

[14] C. Zhang, J. Bi, Y. Zhou, and J. Wu, "Hypervdp: High-performance virtualization of the programmable data plane," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 556–569, 2019.

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

[16] M. Savi, M. Tornatore, and G. Verticale, "Impact of processing-resource sharing on the placement of chained virtual network functions," *IEEE Transactions on Cloud Computing*, 2019.

[17] 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.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, p. 69–74, Mar. 2008. [Online]. Available: https://doi.org/10.1145/1355734.1355746

[19] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.

[20] P4 Community, "P4C Compiler Repository," online at: https://github.com/p4lang/p4c.

[21] W. Tu, F. Ruffy, and M. Budiu, "P4c-xdp: Programming the linux kernel forwarding plane using p4," in *Linux Plumbers Conference*, 2018.

[22] "DPDK Software Switch (SWX) Pipeline Documentation," online at: https://doc.dpdk.org/guides/prog_guide/packet_framework.html#the-software-switch-swx-pipeline.

[23] "Intel P4Studio," online at: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/p4-suite/p4-studio.html.

[24] S. Ibanez, G. Brebner, N. McKeown, and N. Zilberman, "The P4->netFPGA workflow for line-rate packet processing," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 1–9.

[25] "Programming NFP with P4 and C," online at: https://www.netronome.com/media/redactor_files/WP_Programming_with_P4_and_C.pdf.

[26] ETSI, "Network Functions Virtualisation (NFV); Architectural Framework, ETSI GS NFV 002 v1. 2.1," https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf, 2014, accessed: 2021–07-04.

[27] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir, "Segment routing architecture," Internet Requests for Comments, RFC Editor, RFC 8402, July 2018.

[28] C. Filsfils, P. Camarillo, J. Leddy, daniel.voyer@bell.ca, S. Matsushima, and Z. Li, "Srv6 network programming," Working Draft, IETF Secretariat, Internet-Draft draft-filsfils-spring-srv6-network-programming-07, February 2019.

[29] VNF Offloading and Decomposition Framework Repository, online at: https://github.com/ANTLab-polimi/VNF-offloading-framework.

[30] A. Bas, "Leveraging stratum and tofino fast refresh for software upgrades," https://opennetworking.org/wp-content/uploads/2018/12/Tofino_Fast_Refresh.pdf, 2018, accessed: 2021–07-04.

[31] D. Moro, G. Verticale, and A. Capone, "A framework for network function decomposition and deployment," in *2020 16th International Conference on the Design of Reliable Communication Networks DRCN 2020*, 2020, pp. 1–6.

[32] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *Selected Areas in Communications, IEEE Journal on*, vol. 29, no. 9, pp. 1765 –1775, october 2011.