A Framework for Network Function Decomposition and Deployment

Daniele Moro, Giacomo Verticale, Antonio Capone
Dipartimento di Elettronica, Informatica e Bioingegneria
Politecnico di Milano – Italy
{daniele.moro, giacomo.verticale, antonio.capone}@polimi.it,

***Abstract*—Network Function Virtualization (NFV) enables fast provisioning of packet processing logic on general purpose CPUs. This approach, however, does not scale well to very high speed traffic. Programmable hardware solutions, including those based on programmable switches, are emerging as an option for accelerating and scaling network functions. Unfortunately, every type of programmable hardware has specific characteristics that do not make it suitable for running all possible functions. We argue that an efficient strategy is decomposing network functions into components that can run on CPUs or that can be offloaded to specific programmable hardware depending on their characteristics.**

This paper presents a preliminary work on a framework for automating the decomposition and deployment of network functions. The framework includes an orchestrator that chooses the best decomposition according to the traffic demands, the network topology and other constraints. It also provides a tool to combine multiple functions into a single P4 program that can be deployed to a programmable switch. Finally, the framework comprises a set of tools to deploy the network functions either as containers running in a data center or as programs loaded in a programmable switch.

We present numerical results to highlight the advantages of partially offloading decomposed VNFs to programmable hardware over a pure software solution. We also highlight the robustness of the approach showing how the model reacts in case of network failures.

I. Introduction

Network Function Virtualization (NFV) enables fast deployment of functions on general purpose hardware, and can the reduce costs of network provisioning. This approach, however, cannot be easily adopted when Network Functions (NFs) need to process very high traffic loads in the order of hundreds or thousands of gigabits per second. A straightforward solution for processing packets at high speed is horizontally scaling on multiple CPUs. However, this requires to add load balancers that can generate packet reordering, non-deterministic latency and, in general, the inability to work at wire speed. Power consumption may also become an issue when processing high volumes of packets on general purpose CPUs.

While NFV is great for complex but low throughput NFs, like for example those of the control plane, its use can be extremely challenging with data plane NFs. These functions usually perform simpler tasks, but require much higher throughput with respect to control plane functions. A possible option to accelerate this kind of NFs is using programmable network hardware. This makes working at wire speed easier but at the price of limitations in terms of the complexity and the number of operations that can be implemented.

This paper stems from the observation that monolithic implementations are suitable for low-throughput Virtual NFs (VNFs) that can be deployed as software components. On the other hand, when VNFs need to process high traffic, they can be decomposed into multiple, smaller functions, called μVNFs and deployed by distributing these smaller functions onto multiple physical nodes supporting the required operations. Such nodes can be programmable switches, programmable Network Interface Cards (NICs), or in-network compute nodes deployed in micro data centers according to the edge/fog computing paradigm.

This paper provides the following contributions. We present an optimization algorithm for selecting, among multiple possible function decompositions, the one with minimum cost. The algorithm also identifies on which node each μVNF must be deployed. We also present an orchestrator capable of deploying the μVNF onto an SDN network comprising both hardware programmable using the P4 language [1] and a software running a virtual P4 switch in a container. Finally, we present a tool for combining multiple μVNFs into a single P4 program that can be instantiated on a programmable switch. We provide numerical results showing how deploying programmable network hardware and decomposed VNFs can be beneficial with respect to pure monolithic software solutions. We also investigate the robustness of the algorithm to network failures showing how the model reacts to link removal.

The paper is structured as follows. Section II describes the state of the art in NFV decomposition and placement and reports on similar efforts in the literature. Section III describes the scenario considered and our assumptions. Section IV describes the optimization model used to identify and route the best VNF decomposition. Section V discusses the tools we developed to perform the deployment of the solution and the consolidation of different μVNFs into the same programmable switch. Section VI reports our numerical evaluation of the advantages of a mixed hardware/software solution. Finally, Section VII provides some concluding remarks.

II. Related Work

Examples of packet processing logic offloading to programmable hardware have been explored in the literature in recent years. Some works propose to offload the connection

tracking logic in order to reduce the load of DPI engines [2]. Others propose to use programmable hardware for offloading load balancing logic [3], managing key-value stores [4], or accelerating Linux iptables [5]. More complex scenarios also consider the coordination of multiple nodes [6], and MapReduce acceleration [7]. This paper generalizes those ideas to the case of general VNFs. Additionally, we propose an optimization model for deciding whether it is best to deploy a specific function as a software component or as a program in a programmable network hardware.

The problem of optimal deployment of VNFs has also attracted researchers' attention in recent years. The authors of [8] were among the first to jointly study optimal placement and routing of VNFs, while the authors of [9] describe a linear model that takes into account the cost of scaling a function to multiple CPUs. In [10], the authors propose a model for choosing among multiple possible VNF functional decompositions, but do not consider programmable hardware as a possible target for deployment. In this work, in addition to what has been done in those papers, we propose a model that considers a mixed hardware/software scenario with different constraints and costs depending on the type of node.

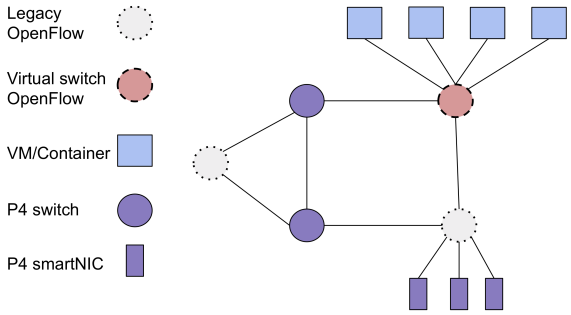

Fig. 1. Example of network with nodes of different types. The group of nodes in the upper right corner represents a data center that comprises a virtual switch and a set of VM/container nodes.

III. System Model

We consider a network with different types of node. Figure 1 shows an example of such network. In particular we consider:

- Nodes with P4 programmable hardware. These nodes can be programmable switches or servers equipped with smart NICs. Each node is characterized by a maximum width for the programmable pipeline (i.e. the number of pipeline stages executed one after the other), a maximum depth for the programmable pipeline (i.e. number of entries that can be installed in a table), and by a set of boolean flags indicating whether the device provides specific functions (`extern` function in P4 language).
- OpenFlow switches. These nodes are switches that can be controlled via the OpenFlow protocol, they are mainly used as traffic forwarding nodes.
- Nodes that support the dynamic creation of P4 programmable virtual switches (e.g. in a container). These virtual switches support a pipeline of unlimited width and length and provide all the possible `extern` functions. Each of these nodes also includes a virtual OpenFlow switch for routing traffic among containers and with the physical interfaces.

In addition, the network includes an ONOS SDN controller [11], which is responsible of configuring all the switches to route the traffic end-to-end. ONOS also deploys the flows both on the pure OpenFlow switches and on the P4 programmable switches.

Finally, an orchestrator receives the network topology and the traffic demands for the network functions. Then, it chooses the optimal decomposition, its routing, and instructs ONOS to deploy the function code and populate the flow tables.

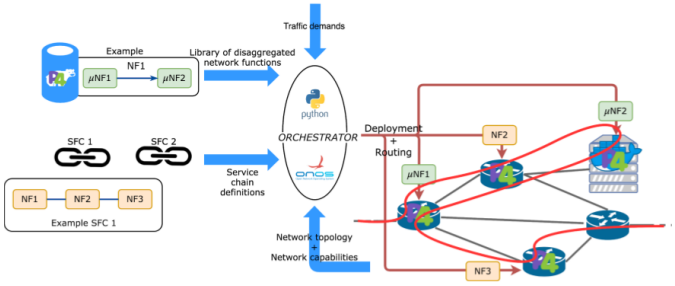

Fig. 2. The components of the proposed framework.

Figure 2 shows the components of the framework. Different NFs can be combined to form a Service Function Chain (SFC). The implementation of the various network functions is stored in a catalog. For the sake of simplicity, we assume that the code for each function is available in the P4 language. The catalog also contains, for each function, several possible decompositions into multiple μVNFs. Some of these could be additionally decomposed into smaller μVNFs. Additionally, for each μVNF, the catalog may contain multiple implementations with different requirements in terms of width and depth of the pipeline and in terms of `extern` functions available in the switch. Such decompositions and implementations could even be generated on-the-fly from a single source. Figure 3 shows an example. In this paper, we assume that multiple decompositions and implementations are pre-generated and stored in the catalog.

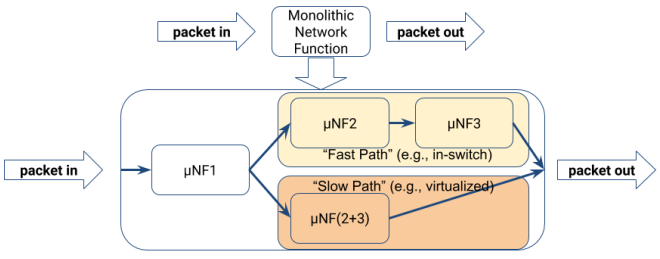

Fig. 3. Example of VNF decomposition. A single monolithic network function can be decomposed in several smaller network functions.

IV. Optimization Model

We propose the following mixed integer linear optimization model to choose the optimal decomposition of a single SFC, the positioning of the µVNFs, and the routing of the traffic through the µVNFs of the chosen decomposition.

a) Sets:

VNF: Set of all the possible VNFs and µVNFs

DC: Set of all the possible decompositions

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

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

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

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

C^{dc}: The virtual graph
$C^{dc} = (VN_{dc}, G_{dc})$

$DST \in U_{HOST}$ Destination of the SFC

$SRC \in U_{HOST}$ Source 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 of the link (u, v)

$extern_i$ Set of P4 "extern" function required by VNF i

ex_u Set of P4 "extern" function 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.

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

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

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

d) Objective function:

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

with

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

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

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

e) Decomposition constraints:

$$\sum_{u \in U} x_{dc,i,u} = z_{dc} \quad \forall dc \in DC, i \in VN_{dc} \tag{5}$$

$$x_{dc,S_{dc},SRC} = z_{dc} \quad \forall dc \in DC \tag{6}$$

$$x_{dc,D_{dc},DST} = z_{dc} \quad \forall dc \in DC \tag{7}$$

$$\sum_{u \in U, u \neq SRC} x_{dc,S_{dc},u} = 0 \quad \forall dc \in DC \tag{8}$$

$$\sum_{u \in U, u \neq DST} x_{dc,D_{dc},u} = 0 \quad \forall dc \in DC \tag{9}$$

f) Constraints on node types: Only P4 programmable hardware nodes and software nodes can allocate functions, while standard OpenFlow devices and hosts can not.

$$\sum_{\substack{dc \in DC \\ i \in VN_{dc} \\ u \in U_H \cup U_{OF}}} x_{dc,i,u} = 0 \quad \forall dc \in DC \tag{10}$$

g) Capacity constraints:

$$x_{dc,i,u} \leq 1 \quad \forall dc \in DC \quad \forall (i, u) \in \{(i, u) | ex_i \subseteq extern_u, i \in VN_{dc}, dc \in DC, u \in U\} \tag{11}$$

$$\sum_{\substack{dc \in DC \\ i \in VN_{dc}}} depth_i \cdot x_{dc,i,u} \leq d_u \quad \forall u \in U_{hw} \cup U_{sw} \tag{12}$$

$$\sum_{\substack{dc \in DC \\ i \in VN_{dc}}} width_i \cdot x_{dc,i,u} \leq w_u \quad \forall u \in U_{hw} \cup U_{sw} \tag{13}$$

$$\sum_{\substack{dc \in DC \\ i \in VN_{dc}}} depth_i \cdot tr \cdot x_{dc,i,u} \leq CPU_u \quad \forall u \in U_{sw} \tag{14}$$

h) Link to path mapping:

$$\sum_{e_{u,v} \in E} f_{dc,e_{i,j},e_{u,v}} - \sum_{e_{v,u} \in E} f_{dc,e_{i,j},e_{v,u}} = x_{dc,i,u} - x_{dc,j,u}$$
$$\forall dc \in DC, i \in VN_{dc}, u \in U \quad (15)$$

i) Bandwidth constraint:

$$\sum_{\substack{dc \in DC \\ e_{i,j} \in G_{dc}}} tr \cdot f_{dc,e_{i,j},e_{u,v}} \leq BW_{e_{u,v}} \forall e_{u,v} \in E \quad (16)$$

V. Deployment

The orchestrator is in charge of deploying the allocated network functions and routing the traffic accordingly.

From the solution of the optimization model, the orchestrator identifies the nodes where µVNFs must be deployed. The SFC is then routed through these waypoints using Segment Routing version 6 (SRv6) [12]. Each SRv6 segment is a 128-bit address called Segment Identifier (SID). SIDs are used to identify nodes and network functions together in the network. In particular, the 64 most significant bits are used to identify the location that the packet has to reach (e.g., physical or software node), while the 64 least significant bits are used to identify the NF that needs to be executed. This usage of SRv6 is in line with the SRv6 Network Programming Internet draft [13]. The orchestrator generates the SIDs that every packet of the specific SFC has to carry, and the flow rules that are installed in each node of the network.

The function deployment phase includes also the dynamic generation of P4 programs that are installed on all the nodes that support P4 (e.g. hardware programmable nodes and virtual switches running on general purpose hardware). Functions such as the parsing of SRv6 packets and their routing and forwarding are common to all the nodes. For this reason, we define a template pipeline into which the orchestrator plugs the specific network functions. The orchestrator retrieves the network functions from the catalog, composes them and creates the final P4 program that is deployed on the network nodes.

A standard P4 program is composed of a parser, a packet processing stage and a deparser. The template pipeline comprises a parser and a deparser, which are the same for all the nodes in the network. The parser support IPv6 packets with SRv6 headers, while the pipeline is composed of the 3 high-level stages. Figure 4 shows an high level scheme of the pipeline.

The first stage is dedicated to SRv6 processing. If the received packet has an SRv6 header, the next SID is extracted using the `Segments Left` header field. This SID is used to override the IPv6 address when the packet is forwarded. Then, a match on the 64 most significant bits of the IPv6 destination address is performed. If the match is positive, the local node is the actual destination of the packet, and the network function identifier is extracted from the 64 least significant bits of the matched address. The IPv6 destination address is also overridden to set the next hop. Finally, the next hop table is matched to understand on which port the packet has to be sent out.

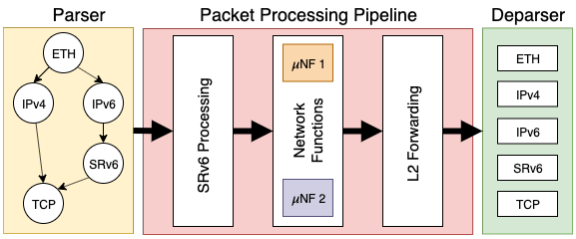

Fig. 4. Pipeline of the P4 programmable switch supporting µVNFs deployment.

The second stage of the pipeline is in charge of executing the actual network function. If, in the previous stage, a network function identifier address has been set, then a dispatcher will execute the correspondent NF. The network function can override the next hop, modify the packet, or perform other packet processing functionalities. The state can also be transferred from a function deployed in one node to the next or to the following ones. In particular, each network function can read and write SRv6 metadata. These metadata are parsed from the packet received and are deparsed and sent to the next node in the service chain.

The last stage is dedicated to packet forwarding and layer 2 operations. If the current node is not the destination of the packet and no network function has been executed by the node itself, the packet is forwarded according to layer 2 information.

The orchestrator generates the P4 programs for the involved nodes, compiles each of them for the specific hardware, and then deploys the resulting compiled programs on the selected network nodes. If the node is a hardware device, the compiler output is directly installed on it. If the node is a software node, the orchestrator spawns a new P4 virtual switch on the general purpose computing node and connects it to the local OpenFlow switch. Then, it deploys the compiled program. Finally, the routing information is pushed to ONOS, which, in turn, installs the corresponding flow rules into the network nodes.

VI. Numerical Results

In this Section we report on the performance of the optimization algorithm over a few relevant architectures. We also report on the robustness of the optimization algorithm to network failures. Numerical results are based on simulations. The simulation topology is the *BT-Europe* from the Internet Topology Zoo[14]. We generate multiple random instances and solve the optimization problem using the Gurobi solver.

Four scenarios are considered. The first one, named *DC scenario*, comprises a single, large data center in which all the SFCs and all the VNFs are deployed. For this scenario, we select a single node at random from the topology as the data center. The data center node has 50,000 units of computation capacity. The second scenario, named *µDC scenario*, is a network composed of multiple smaller data centers. One third of the nodes have computation capabilities and can allocate

functions. For these two scenarios, all the other nodes are considered simple OpenFlow devices that can only forward traffic. Furthermore, these OpenFlow devices have attached a host that can be the traffic source for an SFC.

The following two scenarios enrich the previous two with P4 programmable hardware (smartNICs and switches). In particular, in the *P4-DC scenario*, we still have a single data center, but half of the other nodes are P4 programmable hardware. In the *P4-µDC scenario*, we consider a third of nodes having a co-located micro data center and half of the nodes having programmable hardware. For the two scenarios with programmable hardware, we consider two different kinds of hardware. We consider smartNICs, which have a pipeline depth of 20 and a pipeline width of 10, and switches, which have a pipeline depth of 100 and a pipeline width of 50. For each node equipped with P4 hardware, the probability of being a switch is double with respect to smartNICs.

We run the simulations with 150 consecutive SFC requests, taken from a pool of 10 randomly generated SFCs. Each SFC requests an amount of traffic between 1 and 50. The number of network functions within a single SFC is uniformly distributed between 1 and 10. Each VNF can be implemented with different decompositions, which is a number between 2 and 5 with uniform distribution. The number of µVNFs composing a VNF is a uniformly generated number between 1 and 5. Each µVNF is characterized by a pipeline occupancy in depth and width; these parameters are generated at random between 1 and 15, and 1 and 20 respectively.

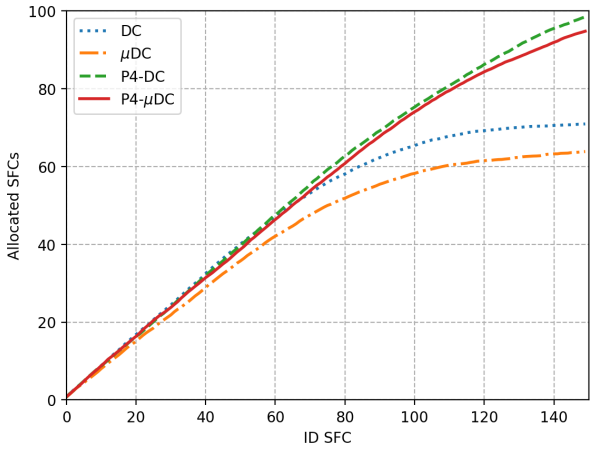

Fig. 5. Number of accepted SFCs vs number of offered SFCs for the four configuration options.

Figure 5 shows how the accepted number of SFCs grows as the number of offered SFCs grows. Similarly, Figure 6 shows, for the same scenarios, how the total cost of the accepted SFCs grows as the number of offered SFCs grows. From the Figure 5 we can see that the solutions that comprise programmable hardware accept more SFCs than the ones with only data centers. Figure 6 makes possible to get some additional insights. The solutions with no programmable hardware have a cost that grows linearly with the number of offered SFCs. Instead, the

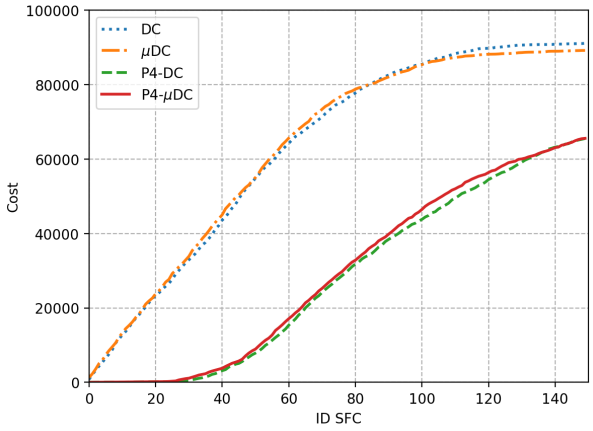

Fig. 6. Total cost of the accepted SFCs vs number of offered SFCs for the four configuration options.

programmable hardware is able to accommodate a significant number of SFCs at little cost thus increasing the overall capacity. It is also worth noting that the µDC solution has worst performance than the solution with a single data center. In particular, while the total cost of deployment grows at the same rate, the µDC starts dropping SFCs earlier, resulting in a lower capacity. In the scenario with programmable hardware there is no difference between having a single DC or multiple smaller µDCs: the cost grows at a similar rate and the two solutions also start dropping SFCs at the same load.

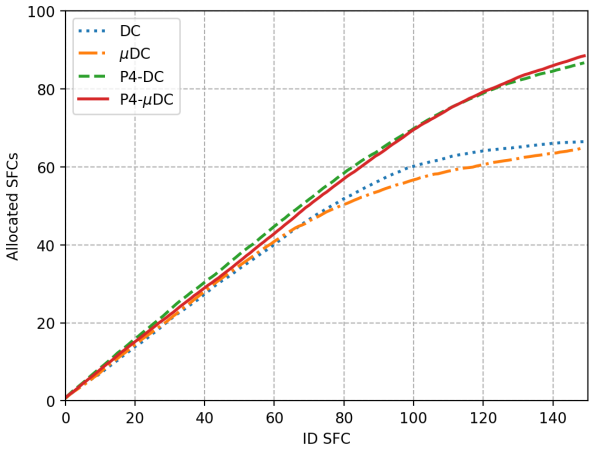

Fig. 7. Number of accepted SFCs vs number of offered SFCs for the four configuration options after removal of 1 link.

We also want to compare the robustness of the various deployment scenarios to network failures. Thus, we evaluate how the network capacity, in terms of deployable NFs, changes when one or two links are removed. Figures 7 and 8 show the number of accepted SFCs vs the number of offered SFCs in the network with one random link removed and two random links removed, respectively. The figures show that the solutions with µDCs have the least capacity loss both in the scenario without programmable hardware and in the scenario

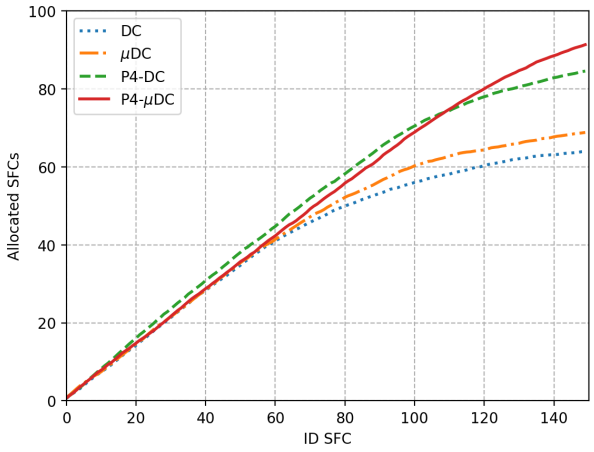

Fig. 8. Number of accepted SFCs vs number of offered SFCs for the four configuration options after removal of 2 links.

with programmable hardware. Additionally, we note that the advantage of programmable hardware generally diminishes as more links are removed.

VII. Conclusion

In this paper, we present a framework for decomposing a network function (NF) in smaller units, called μVNFs, that can be deployed in a distributed fashion. Each μVNF can be deployed either as a software program executed on general purpose CPUs or can be merged with other μVNFs and deployed on programmable hardware. We propose an algorithm for choosing the optimal decomposition of the NF, placement, and the routing of the μVNFs. Finally, we also present a toolset, based on the ONOS SDN controller, for deploying the functions. Routing of packets from one μVNF to the following and selection of the μVNF in a node exploits the framework for segment routing currently under standardization by the IETF.

Results show that the availability of programmable hardware in the network increases the network capacity in terms of the number of functions that can be deployed and also makes the network more robust. Future work will focus on the improvement of the optimization model to include function consolidation exploiting μVNFs already deployed in the network and to add new constraints for taking into account link and node delays. We will also work on the development of heuristics for online allocation integrating them into the proposed framework.

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] 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.

[4] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 121–136.

[5] 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.

[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] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.