Distributed Service Function Chaining

Milad Ghaznavi, Nashid Shahriar, Shahin Kamali, Reaz Ahmed, and Raouf Boutaba, *Fellow, IEEE*

***Abstract*— A service-function chain, or simply a chain, is an ordered sequence of service functions, e.g., firewalls and load balancers, composing a service. A chain deployment involves *selecting* and instantiating a number of virtual network functions (VNFs), i.e., softwarized service functions, *placing* VNF instances, and *routing* traffic through them. In the current optimization-models of a chain deployment, the instances of the same function are assumed to be identical, while typical service providers offer VNFs with heterogeneous throughput and resource configurations. The VNF instances of the same function are installed in a single physical machine, which limits a chain to the throughput of a few instances that can be installed in one physical machine. Furthermore, the *selection*, *placement*, and *routing* problems are solved in isolation. We present distributed service function chaining that coordinates these operations, places VNF-instances of the same function *distributedly*, and selects appropriate instances from typical VNF offerings. Such a deployment uses network resources more efficiently and decouples a chain's throughput from that of physical machines. We formulate this deployment as a mixed integer programming (MIP) model, prove its NP-Hardness, and develop a local search heuristic called Kariz. Extensive experiments demonstrate that Kariz achieves a competitive acceptance-ratio of 76%–100% with an extra cost of less than 24% compared with the MIP model.**

***Index Terms*— Service function chaining, network function virtualization, virtual network function, placement, routing.**

I. Introduction

A SERVICE-FUNCTION chain, or simply a *chain*, is an ordered sequence of service-functions composing a service [37]. For example in a typical data-center network, traffic from a server passes through an IDS, a firewall, and a NAT before reaching to the Internet [5]. Until recently, functions have been vertically integrated in dedicated hardware middleboxes, i.e., a chain of pricey middleboxes are provisioned to provide throughput for peak-load, and traffic must be routed through fixed locations in which these middleboxes are placed [36].

Network function virtualization decouples network functions from underlying hardware and implements them as software appliances on virtualized commodity hardware. These appliances are called Virtual Network Functions (VNFs). In this way, a chain of inexpensive VNFs provide same packet-processing functions at a desired throughput, and we can route traffic through appropriate locations in which VNF-instances are dynamically placed. Such a deployment reduces capital and operational costs and optimizes network operations.

A chain deployment involves *selecting* and instantiating a number of VNFs, *placing* these VNF-instances, and *routing* traffic through them. An optimal chain deployment *coordinates* the *selection*, *placement*, and *routing* to minimize resources allocated while satisfies the resource capacity and location constraints. Existing VNF chaining models have several limitations as follows.

A. Gaps in Selection

Most of the optimization models [8], [10], [45] do not consider the typical VNF offerings and assume that VNFs of a same function are identical in their resource consumption and throughput. Service providers offer VNFs with different configurations to provide predictable quality of service. For example, HP offers virtual IPSec [3] that provides throughputs of 268, 580, and 926 Mbps assuming respectively 1, 4, and 8 CPU cores. Similarly, Riverbed offers WAN-optimizers [6] with throughputs of 10 and 50 Mbps given respectively 2 and 4 CPU cores. Note that the correlation between the throughput and resource consumption is not necessarily linear. In practice, predicting the performance of service-functions is not trivial [9], [13], [25].

B. Gaps in Placement and Routing

To process a traffic flow, some models use a single physical-machine to place VNFs of a same function [8], [10], [27], [29], [32] or even all VNFs of a chain [39]. However, doing so severely limits throughput of the function and chain to a few VNFs that can be instantiated in a physical-machine. The throughput of these instances might not be sufficient to process the total traffic routed through them, and this problem is exacerbated by the fact that traffic volume through functions has an increasing trend [23], [47].

C. Gaps in Coordination

A VNF cannot be selected if there is not sufficient resources to place its instances. Further, it is impractical to place an instance in a given location when adequate bandwidth is not available to route traffic from/to the location. To achieve an optimal deployment of service chains, selection, placement, and routing must be performed in a coordinated manner; otherwise, the deployment results in sub-optimal utilization of network resources and quality of service. Most of existing solutions solve the placement and routing in isolation [15], [17], [36], [46]. There are few solutions [10], [32]

Manuscript received April 1, 2017; revised September 12, 2017; accepted September 25, 2017. Date of publication October 5, 2017; date of current version December 1, 2017. *(Corresponding author: Raouf Boutaba.)*

M. Ghaznavi, N. Shahriar, R. Ahmed, and R. Boutaba are with the University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: eghaznav@uwaterloo.ca; nshahria@uwaterloo.ca; r5ahmed@uwaterloo.ca; rboutaba@uwaterloo.ca).

S. Kamali is with the University of Manitoba, Winnipeg, MB R3T 2N2, Canada (e-mail: shahin.kamali@umanitoba.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2017.2760178

that coordinate the placement and routing; however, they treat the selection of VNFs separately.

To fill the above gaps, we present distributed service function chaining (DSFC). For each function of a chain, DSFC selects from provided VNF offerings and determines the appropriate number of VNF-instances to be placed. DSFC places these instances in a way that VNFs of a same function can be installed distributedly in multiple machines. Such a placement decouples a chain's throughput from physical-machines. Further, DSFC, utilizing the global knowledge of the network, routes traffic and distributes the load among the VNF-instances. DSFC coordinates selection, placement, and routing operations in such a way that network resources are utilized more efficiently.

Specifically, our contributions in this paper are: *i)* we model and solve DSFC using mixed integer programming (MIP), and prove its NP-Hardness. *ii)* for larger networks, we propose *Kariz*, a local search heuristic that employs a tuning parameter to balance the speed-accuracy trade-off; *iii)* we perform extensive simulations to evaluate Kariz against the MIP implementation for various chain-lengths and throughput-demands. The results demonstrate that Kariz achieves the competitive acceptance ratio of 76-100% at an extra cost of less than 24%, in comparison to the MIP implementation.

The rest of the paper is organized as follows. In § II, we study related works. § III discusses the system implementation and deployment challenges. We present our problem formulation in § IV. Our solution is proposed and evaluated in § V and § VI, respectively. Lastly, § VII concludes this paper.

II. Related Work

A. Selection

VNF-P [32] studies a hybrid deployment scenario using hardware-middleboxes and VNFs to provide a requested service. VNF-OP [10], [22], JoraNFV [44], and [29] model batch-deployments of multiple chains. Reference [34] is a scheduling framework for deploying VNFs. These papers assume that VNFs of the same function are identical. Slick [8] is a framework that allows users to write fine-grained *elements* to perform custom packet-processing. Predicting the performance of such arbitrary packet-processing element is not trivial. In contrast to these studies, we select appropriate VNFs from different typical offerings providing predictable performance.

B. Placement

Split/Merge [38] and OpenNF [18] redistribute packet-processing across a collection of VNF-instances. In contrast to DSFC, they do not focus on placement optimization models. Stratos [17] orchestrates VNF-instances on a remote cloud. It uses a rather simple technique that places VNFs of a chain as closely as possible to each other. JVP [27] considers the relation of bandwidth usage and host resource usage in the deployment of chains. However, JVP instantiates a single VM for each VNF. VNF-OP [10] and VNF-P [32] place all VNF-instances of a function on a single machine. In contrast to these works, we place multiple VNF-instances of each function distributedly. CoMb [39] is an architecture designed to consolidate the chain deployment. In contrast to DSFC, CoMb places all VNFs of a chain that deal with the same session at a fixed location. [30] only optimize the placement of VNFs and does not consider the routing.

C. Routing

Unlike our work, [15], [36], [46] optimize only bandwidth usage. In processing a network flow, Slick [8] uses a single instance for each function. On the contrary, DSFC routes traffic among multiple VNF-instances for each function. Stratos [17] solves the routing separately after placing VNF instances. LightChain [22] optimizes the number of switches between ingress and egress points of chains. The authors of [19] solve the joint placement and routing problem using a dynamic programming algorithm. E2 [34] instantiates VNFs in certain servers to optimize the inter-server communication. Although [10], [19], [32], [34] coordinate the placement and routing, they still treat the selection separately. We jointly optimize routing, placement, and selection that was not the focus of these studies.

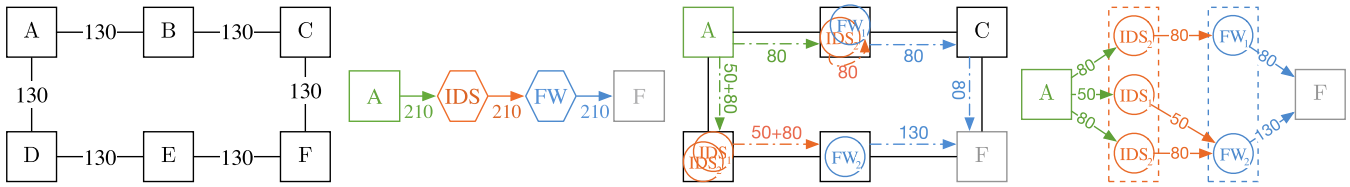
III. Challenges

A chain specifies that the traffic originating from a *source*, is processed by an ordered sequence of functions, and finally is delivered to a *target*. To deploy a chain distributedly, several system and optimization challenges have to be addressed.

A. System Implementation Challenges

Service-functions often operate on data-packets at a *flow* granularity and maintain *state information* on the flows and sessions they process [41], [43]. State information consists of configuration and statistical data, and differs from one function to another. If a function is replaced with multiple VNF-instances, the functionality should not change, and these instances must act in concert. Further, the traffic processed by a single function, should now be processed by multiple VNF-instances. Thus, *consistent state distribution* and *consistent traffic distribution* among the VNF-instances are essential.

1) Consistent State Distribution: Deployment of multiple VNF-instances to provide a function requires distribution of the state information. Hence, we need to *model* the state information and *distribute* it among the VNF-instances consistently. The state information can be classified as *internal* or *external*. The internal state is stored and used only by a single instance, while the external state is distributed and shared across multiple instances. Since the state information is stored in a key-value store [24], [41], data structures like distributed hash-tables and technologies like *remote direct memory access* (RDMA) can fulfill this challenge efficiently. Moreover, it might be required to modify the functions to cope with the defined model. There are abstraction models and system implementations that address this challenge. Rajagopalan *et al.* [38] introduce a system-level abstraction called Split/Merge that stores the internal state exclusively inside each VNF-instance, while the external state is distributed and accessible by other instances. As a proof of concept, they implemented FreeFlow



(a) Network (b) Chain (c) Deployed in Network (d) Logical Representation

Fig. 1. Distributed deployment of a chain.

as a Split/Merge system, and ported Bro IDS [35] inside it. Further, they analyzed and confirmed the compatability of two other functions, i.e. application delivery controller and stateful NAT64. In addition, Joseph and Stoica [24] provide a model to describe different functions. As concrete examples, firewall, NAT and layer4 and layer 7 load-balancer are described using the proposed model. Further, Gember *et al.* [16] and OpenNF [18] introduce a unified framework to manage state information.

2) Consistent Traffic Distribution: Replacing a single function with multiple VNF-instances requires *splitting* and *distributing* the traffic load among these instances. Per-flow traffic splitting distributes the traffic in the granularity of flows, and packets of a flow have to be routed along the same path. Split/Merge [38] utilizes a similar approach. However, this approach does not support accurate load-distribution and is not always applicable. For instance, if the load of a flow is higher than the throughput of an assigned VNF-instance, that instance cannot handle the load and we have to split the traffic into a smaller granularity. *Flowlet switching* [7], [40] can be leveraged to split the traffic into a finer granularity. A flowlet is a "burst of packets from the same flow followed by an idle interval" [40]. If the interval between two flowlets is greater than the maximum delay difference between parallel paths, the second flowlet—and consequently following flowlets—can be sent through different paths. Thus, a single flow can be split into multiple paths without packet-reordering. Furthermore, accurate load balancing is achieved using short flowlet intervals ($[50, 100]ms$) [40]. Specifically, flowlets are abundant in data-center networks since the latency is very low and the traffic is intensively bursty [26]. In addition to these distributed methods, the central schemes leveraging SDN and OpenFlow capabilities [28] can also be used. For instance, *group tables* [4] can be used to split and balance the traffic.

We have shown the feasibility of distributed deployment of VNF-instances to provide a function and distributing traffic among these instances. Next, we state the assumptions that ground our optimization model:

- The state information of a function can be consistently distributed among multiple VNF-instances. This assumption holds for the state information of a single flow.
- The traffic can be consistently distributed into multiple paths among multiple VNF-instances. This assumption holds for a single flow.

B. Optimization Challenges

The optimization challenge is in computing an optimal allocation of host and bandwidth resources to a chain.

TABLE I
VNFs

Function	VNF	Throughput	CPU demand	Memory demand
IDS	IDS_1	50 Mbps	1 core	24 GB
	IDS_2	80 Mbps	1 core	32 GB
Firewall	FW_1	100 Mbps	1 core	1.75 GB
	FW_2	200 Mbps	2 core	3.50 G

Each function in a chain is replaced with a number of VNF-instances providing the requested throughput. These instances are placed in a set of chosen hosts. In addition, the traffic is split and routed among the instances. Thus, certain decisions have to be made optimally: *number of VNF-instances* (*selection*), *placement* of these instances, and *routing the traffic* through the placed instances. These decisions are interdependent and must be made in a coordinated manner.

Fig. 1 shows a chain deployment. The network of Fig. 1a includes 6 hosts, each with an 8-core CPU and 64 GB residual memory. For simplicity, switches are not shown, and the presented paths are disjoint in this example. All paths have 130 Mbps available bandwidth. The chain of Fig. 1b includes 2 functions with 210 Mbps throughput: an IDS and a firewall (FW). The flow comes from the host A, the source, is processed by IDS and FW, and then sent to host F, the target. As listed in Table I, there are 4 VNF types for IDS and FW. Fig. 1c depicts the chain deployed in the network, and Fig. 1d shows the logical representation of this deployment: with 3 instances for IDS ($1\times IDS_1 + 2\times IDS_2$) and 2 instances for FW ($1\times FW_1 + 1\times FW_2$). The IDS instances are installed in hosts B and D. The flow splits, and 80 Mbps and 130 Mbps are routed from the source to hosts B and D, respectively. FW instances are installed in hosts B and E. In host B, the flow after being processed by IDS_2 is sent to FW_1. IDS_1 and IDS_2 forward the flow to host C in which instance FW_2 is placed. Finally, the flow from the FW instances is sent to the target. Note that it is possible to place the VNF-instances in the source and target if sufficient host resources are available.

IV. Distributed Service Function Chaining

With the assumptions and challenges established, we now introduce the formal definitions and the mathematical model. Table II lists important notations used in this paper.

A. Definitions

1) Physical Resources: $R = \{\text{CPU, memory, storage, } \ldots\}$ represents a set of available physical resources.

2) Network: Graph $G = (N, E)$ is the substrate network, where N and E are substrate nodes and links, respectively. $c_{mr} \in \mathbb{R}^+$ is the residual capacity of node m for

TABLE II
NOTATION

Symbol	Meaning
r	A resource
N	Substrate nodes
n	A substrate node
c_{mr}	Residual capacity of node m for resource r
E	Substrate Links
(m,n)	A substrate link between nodes m and n
c_{mn}	Residual bandwidth capacity of link (m,n)
$\overline{N}$	Virtual nodes
$\overline{s}$	Source (virtual node) of traffic of service chain request
s	Source (substrate node) of traffic of service chain request
$\overline{t}$	Target (virtual node) of traffic of service chain request
t	Target (substrate node) of traffic of service chain request
$\overline{V}$	Virtual functions
$\overline{v}$	A virtual function
$\overline{b}$	Throughput demand of a chain
V	VNFs
u	A VNF
q_u	Throughput of VNF u
d_{ur}	Demand of VNF u for resource r
$x_{mn}^{\overline{u}}$	The volume of traffic type $\overline{u}$ in substrate link (m,n)
y_{mu}	The number of the instances of VNFs u in substrate node n
$z_{m\overline{u}}$	The allocated throughput of VNFs of type $\overline{u}$ in substrate node m
$B(.)$	Bandwidth allocation cost
$H(.)$	Host allocation cost
$L(\overline{u})$	Layer of virtual function $\overline{u}$

resource $r \in R$. Set E_m denotes incident links on node m. Moreover, $mn \in E$ is the link between node $m \in N$ and node $n \in N$ and has a residual bandwidth capacity of $c_{mn} \in \mathbb{R}^+$.

3) Chain: Symbols with over-line are for chain definitions. Forwarding graph $\overline{G} = (\overline{N}, \overline{A})$ denotes a chain. $\overline{N}$ includes functions $\overline{V} \subset \overline{N}$, and two endpoints $\overline{s}$ and $\overline{t}$. Traffic flow coming from $\overline{s} \in \overline{N}$ is processed by functions in the chain, and is forwarded to $\overline{t} \in \overline{N}$. Respectively, $\overline{s}$ and $\overline{t}$ are the *source* and *target* of the traffic. The corresponding substrate nodes for source and target are respectively $s \in N$ and $t \in N$. Function $\overline{v} = f(\overline{u})$ follows function $\overline{u}$. We define *ring* $\overline{uv} \in \overline{A}$ as 2 consecutive functions $\overline{u}$ and $\overline{v}$, where $\overline{v} = f(\overline{u})$. We assume that $\overline{u}$ generates traffic type $\overline{u}$ and $\overline{v}$ consumes this traffic type. Each ring $\overline{uv}$ has the *throughput demand* $\overline{b}$ denoting integer traffic volume flow generated or consumed by the ring nodes.

4) VNFs: Set V denotes VNFs. A VNF $u \in V$ has throughput $q_u \in \mathbb{R}^+$ showing the maximum traffic that u can process. $d_{ur} \in \mathbb{R}^+$ is the demand of u for resource r. These demands include the overhead of accessing distributed state information. For $\overline{s}$ and $\overline{t}$, we assume that there are VNFs $u_{\overline{s}}$ and $u_{\overline{t}}$, respectively. These VNFs have throughput $\overline{b}$ and no demand for any resource. Finally, VNFs of type $\overline{u}$ are identified by $V_{\overline{u}}$.

B. Mathematical Model

Variable $x_{mn}^{\overline{u}} \in \mathbb{R}$ is the traffic volume of type $\overline{u} \in \overline{N}/\{\overline{t}\}$ on substrate link mn. Target $\overline{t}$ is excluded since it only consumes traffic; thus, no traffic of this type exists in the network. Variable $y_{mu} \in \mathbb{Z}$ is the number of instances of VNF u in substrate node m. VNF instances of $V_{\overline{u}}$ installed in node m provide throughput of type $\overline{u}$. Variable $z_{m\overline{u}} \in \mathbb{R}$ denotes the allocated throughput of these VNF instances. A solution for the problem is shown by a tuple of allocation vectors (X, Y, Z), defined as follows. Let vector $X_{\overline{u}} = \{x_{mn}^{\overline{u}} : \forall mn \in E\}$ be allocated bandwidth of links to traffic type $\overline{u}$, and $X = \bigcup_{\overline{u} \in \overline{N}/\{\overline{t}\}} X_{\overline{u}}$. If $Y_{\overline{u}} = \{y_{mu} : \forall m \in N, \forall u \in V_{\overline{u}}\}$ identifies the VNF instantiated for function $\overline{u}$, let $Y = \bigcup_{\overline{u} \in \overline{N}} Y_{\overline{u}}$. Finally, $Z_{\overline{u}} = \{z_{m\overline{u}} : \forall m \in N\}$ denotes the allocated throughput of type $\overline{u}$ in every node, and $Z = \bigcup_{\overline{u} \in \overline{N}} Z_{\overline{u}}$.

1) Node Capacity Constraint: Eq. 1 ensures that instances are placed with respect to the substrate nodes capacities.

$$\forall m \in N : \forall r \in R : \sum_{u \in V} y_{mu} d_{ur} \leq c_{mr} \tag{1}$$

2) Location Constraint: Equalities in Eq. 2 ensure that an instance of $u_{\overline{s}}$ and an instance of $u_{\overline{t}}$ are placed only in $s \in N$ and $t \in N$, respectively.

$$\begin{aligned} y_{su_{\overline{s}}} = 1, \quad \sum_{m \in N/\{s\}} y_{mu_{\overline{s}}} = 0 \\ y_{tu_{\overline{t}}} = 1, \quad \sum_{m \in N/\{t\}} y_{mu_{\overline{t}}} = 0 \end{aligned} \tag{2}$$

3) Substrate Link Capacity Constraint: Eq. 3 makes sure that the capacities of substrate links are not violated.

$$\forall mn \in E, m < n : \sum_{\overline{u} \in \overline{N}} (x_{mn}^{\overline{u}} + x_{nm}^{\overline{u}}) \leq c_{mn} \tag{3}$$

4) Throughput Constraint: Eq. 4 ensures that the aggregate throughput capacity of instances of type $\overline{u}$ placed in substrate node m is more than allocated throughput $z_{m\overline{u}}$.

$$\forall m \in N : \forall \overline{u} \in \overline{N} : \sum_{u \in V_{\overline{u}}} y_{mu} q_u \geq z_{m\overline{u}} \tag{4}$$

5) Throughput Demand Constraint: Eq. 5 guarantees that for each function $\overline{u}$, throughput $\overline{b}$ is allocated by instances $V_{\overline{u}}$.

$$\forall \overline{u} \in \overline{N} : \sum_{m \in N} z_{m\overline{u}} = \overline{b} \tag{5}$$

6) Flow Conservation Constraint: Eq. 6 is a modified version of the flow-conservation constraint [42]. Let us say that in node $m \in N$, VNF instances of types $\overline{u}$ and $\overline{v} = f(\overline{u})$ are installed. Therefore, VNF instances of $V_{\overline{v}}$ locally process a volume of traffic type $\overline{u}$ generated by instances of $V_{\overline{u}}$. This volume is $z_{m\overline{v}}$. Unprocessed traffic volume should exit the node m. This constraint ensures this phenomenon.

$$\begin{aligned} \forall m \in N : \forall \overline{u} \in \overline{N}/\{\overline{t}\} : \overline{v} = f(\overline{u}) : \\ \sum_{mn \in E_m} \left(x_{mn}^{\overline{u}} - x_{nm}^{\overline{u}}\right) = \left(z_{m\overline{u}} - z_{m\overline{v}}\right) \end{aligned} \tag{6}$$

7) Bandwidth Allocation Cost: Eq. 7 is the bandwidth allocation cost. Coefficient $\beta \in \mathbb{R}^+$ identifies the relative importance of bandwidth resources. The communication overhead to access the distributed state information is negligible vs. the actual service traffic volume.

$$B(X) = \beta \sum_{\overline{u} \in \overline{N}/\{\overline{t}\}} \sum_{mn \in E} x_{mn}^{\overline{u}} \tag{7}$$

8) Host Resource Allocation Cost: Eq. 8 is the cost of allocating host resources to place VNF instances. Coefficient $\alpha_r \in \mathbb{R}^+$ is the relative importance of resource $r \in R$.

$$H(Y) = \sum_{u \in V} \sum_{r \in R} \alpha_r d_{ur} y_{mu} \tag{8}$$

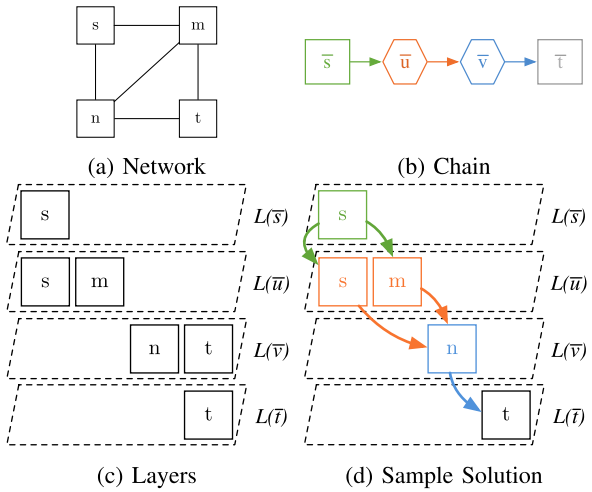


Fig. 2. Layers.

9) Objective Function: Eq. 9 minimizes the aggregate cost of allocating host and bandwidth resources.

$$\min \Big(B(X) + H(Y) \Big) \tag{9}$$

In Appendix A, we use a reduction from the *dominating set problem* to prove our problem, as stated above, is NP-Hard. Our reduction technique is of independent interest and can be potentially applied to analyze the complexity of other problems related to service function chaining.

V. Kariz: Heuristic Solution

Before explaining our solution, we construct a visualization tool to simplify our description. Let us assume that each $\overline{u} \in \overline{N}$ is deployed in a *layer*. Each layer contains a set of nodes in which VNF-instances of a corresponding type can be installed. In other words, in the layer corresponding to $\overline{u}$, we initially place a subset of nodes in which at least a VNF $v \in V_{\overline{u}}$ can be instantiated. $L(\overline{u})$ denotes this layer. Fig. 2c depicts the layers for the chain of Fig. 2b. As shown in Fig. 2c, s and t are the only nodes present in layers $L(\overline{s})$ and $L(\overline{t})$, respectively. Further, nodes $\{s, m\}$ and $\{n, t\}$ are respectively included in layers $L(\overline{u})$ and $L(\overline{v})$ because these nodes have sufficient resources to host VNF-instances of these functions. Fig. 2d presents a sample solution for the chain of Fig. 2b.

Inspired by [21] and [33], we develop a local search heuristic, *Kariz*, which routes traffic layer by layer. We introduce the process first, and then provide a detailed overview. Kariz is shown in Alg. 1 and works as follows. We first initialize layers as described above and set solution as empty (line 1). Starting from layer $L(\overline{s})$ (line 2), iteratively route $\overline{b}$ volume of traffic from layer $S = L(\overline{u})$, the *source-layer*, to the next layer $T = L(\overline{v})$, the *sink-layer* (lines 3-11). After finding the optimal route between two layers (line 5), compute the number of VNF-instances of $V_{\overline{v}}$ by considering the allocated throughput (line 6). Add the solution of the sink-layer to the earlier solution (line 7). Improve the current solution (line 8), and update layers (line 9). Now, traffic has reached the sink-layer; consider this layer as new source-layer (line 10).

Algorithm 1 Kariz Algorithm

1: $init\text{-}layers()$; $(X, Y, Z) \leftarrow (\emptyset, \emptyset, \emptyset)$;
2: $\overline{u} \leftarrow \overline{s}$; $z_{s\overline{s}} \leftarrow \overline{b}$; $z_{t\overline{t}} \leftarrow \overline{b}$; $S \leftarrow L(\overline{s})$;
3: **do**
4: $\quad \overline{v} \leftarrow f(\overline{u})$; $T \leftarrow L(\overline{v})$;
5: $\quad X_{\overline{v}}, Z_{\overline{v}} \leftarrow route(S, T, \overline{b})$;
6: $\quad Y_{\overline{v}} \leftarrow vnf\text{-}instances(Z_{\overline{v}})$;
7: $\quad (X, Y, Z) \leftarrow (X \cup X_{\overline{v}}, Y \cup Y_{\overline{v}}, Z \cup Z_{\overline{v}})$;
8: $\quad improve(X, Y, Z)$;
9: $\quad update\text{-}layers()$;
10: $\quad \overline{u} \leftarrow \overline{v}$; $S \leftarrow L(\overline{v})$;
11: **while** $\big(\overline{u} \neq \overline{t}$ and $S \neq \emptyset\big)$;

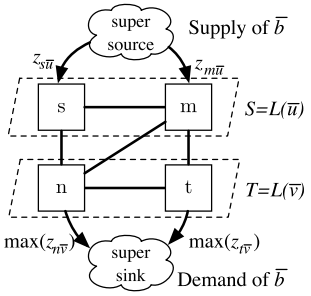


Fig. 3. Routing as single-source single-sink MCFP.

Repeat this procedure if traffic has not reached the last layer yet, and there are nodes in the new source-layer (line 11).

Still to clarify are the traffic routing between two layers and the number of VNF-instances in the sink-layer, how the solution is improved, and how the layers are updated.

A. Route and VNF Instances

Procedure $route(.)$ in Alg. 1 computes the route between two layers by solving the multi-source multi-sink minimum cost flow problem (MCFP) [20]. MCFP is the problem of routing a volume (say $\overline{b}$) of a *commodity* (in our case traffic of type $\overline{u}$) from multiple sources (say a source-layer) to multiple sinks (in our case a sink-layer). Any multi-source multi-sink MCFP can be modeled as a single-source single-sink MCFP that is solvable in polynomial time [20]. For our problem, this is achieved by representing the source- and sink-layers with imaginary nodes *super-source* and *super-sink*, respectively. Fig. 3 depicts this model for layers S and T in Fig. 2. The procedure is as follows. Add a super-source and connect it to every node $m \in S$ in the source-layer with a directed-link whose capacity is $z_{m\overline{u}}$. For the sink-layer, add a super-sink node and connect every node $n \in T$ using a directed-link. The capacity of the directed-link connecting node n to the super-sink is the maximum throughput $\max(z_{n\overline{v}})$ of the VNF-instances that can be installed in node n. There is no cost to sending the traffic via these links. As the result, the minimum cost route of traffic from super-source to super-sink gives the optimal routing between the two layers. If p denotes the super-sink, the throughput allocation in each $n \in L(\overline{v})$ is $z_{n\overline{v}} = x^{\overline{u}}_{np}$.

Finding the capacity of directed-links from the sink-layer to the super-sink is similar to the problem of $vnf\text{-}instances(.)$.

The former is finding the maximum throughput $\max(z_{n\overline{v}})$ out of VNF-instances that can be installed in node n. The latter is finding the minimum allocation of resources to VNF-instances providing throughput of at least $z_{n\overline{v}}$ in each node $n \in L(\overline{v})$. In fact, these problems are *dual* and can be modeled as a *multidimensional knapsack problem* [12]. Think of the node as an $|R|$-*dimensional knapsack*, each *dimension* corresponding to a resource $r \in R$. The *items* to be packed are VNF-instances with the *profits* of their throughputs and *weights* of their host resource demands. Though this problem is known to be NP-Hard [12], since the resources of a single machine, especially the number of CPU cores, are limited, the problem size is small. Thus, we can solve it efficiently. Instead, as CPU cores are the most pricey and restricted resources, a feasible solution optimizing the number of allocated cores is a good optimum.

B. Solution Improvement Rounds

Routing between two layers focuses on the cost of traffic routing and does not consider the cost of host resource allocation. Doing so may lead to high host resource cost. Hence, we need to improve the solution. Procedure $improve(.)$, as shown in Alg. 2, facilitates this: repeatedly search for some *actions* to improve the solution (lines 2-8). If no such action is found, report the current solution (line 4-6). Otherwise, perform the action with the greatest drop in cost, the best *admissible* action (line 7), and continue with the adjusted solution. We define actions and *admissibility* in § V-B1 and § V-B3, respectively.

Algorithm 2 Procedure $improve(.)$

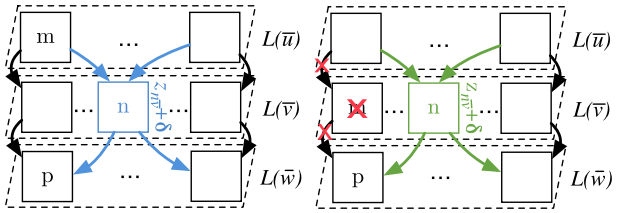
```
1: procedure improve(X, Y, Z)
2:     loop
3:         a ← best-action(X, Y, Z);
4:         if not admissible(a) then
5:             return (X, Y, Z);
6:         end if
7:         perform-action(X, Y, Z, a);
8:     end loop
9: end procedure
```

1) Actions: An action is a *local transformation* intended to reduce the solution's cost. Let (X', Y', Z') be the modified solution after performing an action on a current solution (X, Y, Z). The cost difference before and after performing an action is regarded as the *action cost*, as defined in Eq. 10. The best action has the lowest cost.

$$\big(B(X') + H(Y')\big) - \big(B(X) + H(Y)\big) \tag{10}$$

We define the below actions variants of actions used by [33]:

- $add(n, L(\overline{v}), \delta)$: Include node $n \in N$ in $L(\overline{v})$ and allocate more $\delta > 0$ units of throughput in this node ($z_{n\overline{v}} \leftarrow z_{n\overline{v}} + \delta$). Then, find the minimum cost routing from layer $L(\overline{v})$ to the next and previous layers in the current solution, given allocated throughputs of $L(\overline{v})/\{n\}$. The next and previous layers are $L(\overline{w})$ and $L(\overline{u})$ if $\overline{w} = f(\overline{v})$ and $\overline{v} = f(\overline{u})$, respectively. Finally, tune the allocated throughput of nodes $L(\overline{v})$. This action is shown in Fig. 4a.
- $open(n, M, L(\overline{v}), \delta)$: Add node $n \in N$ into layer $L(\overline{v})$, remove nodes $M \subseteq L(\overline{v})$, and allocate more $\delta > 0$ units of throughput in node n ($z_{n\overline{v}} \leftarrow z_{n\overline{v}} + \delta$). Finally, reroute the traffic either received or originated in layer $L(\overline{v})$. This action replaces a set of fragmented VNFs installed in different nodes M with VNFs collocated in one node n. This action makes sense only if $\delta \geq \sum_{m \in M}(z_{m\overline{v}})$. Fig. 4b depicts an example of this action.



(a) $add(n, L(\overline{v}), \delta)$ (b) $open(n, \{m\}, L(\overline{v}), \delta)$

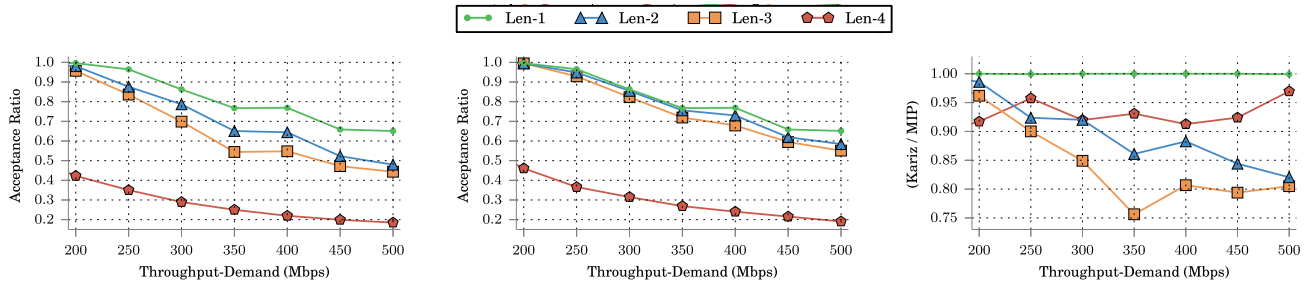
Fig. 4. Actions.

Traffic routing in the above actions is a bit different from routing in $route(.)$. The difference lies in routing two different traffic types. Considering each traffic type as a commodity, still this problem can be modelled it as a multi-commodity MCFP (real flows) that is solvable in polynomial time [14].

We also need to examine actions and select the best in polynomial time and ensure that the number of performed actions is not exponential. Particularly, we need to select the best action with sufficient improvement efficiently. These criteria, *efficient action selection* and *sufficient improvement*, are essential to assure that the algorithm terminates in polynomial time.

2) Efficient Action Selection: The number of $add(.)$ actions is less than $|N| \times |\overline{V}| \times \overline{b}$ under the assumption of integrality of $\overline{b}$. Thus, it is possible to check all actions and select the best in polynomial time. We can even do better and select the value of δ considering the throughputs of VNFs $V_{\overline{v}}$. However, the number of possible $open(., M, L(\overline{v}), .)$ actions is equal to the number of subsets $M \subseteq L(\overline{v})$ which is exponential ($2^{|L(\overline{v})|}$). Thus, we need an efficient procedure to select a good $open(.)$ action. For a fixed layer $L(\overline{v})$, fixed node $n \in N$ and fixed δ, we find this subset in a greedy procedure working as follows. Starting from empty set M, iteratively remove a node m from $L(\overline{v})$ and add it to M. Removing this node has the minimum cost vs. other nodes $L(\overline{v})/m$. Continue this procedure while such a node $m \in L(\overline{v})$ exists, the removal of m decreases the cost, and m's throughput is less than $\delta - \sum_{p \in M} z_{p\overline{v}}$. This procedure repeatedly removes a node $m \in L(\overline{v})$ whose removal results in the greatest decrease in both bandwidth and host resource allocation costs.

3) Sufficient Improvement: If we allow performing actions that yield minor improvements, the number of actions can be large. Thus, only actions with sufficient cost improvement are allowed. An action yielding sufficient improvement is said *admissible*. More precisely, we define an action as admissible if it improves the solution no less than $\frac{\epsilon}{5|N|}\big(B(X)+H(Y)\big)$ for some tuning parameter $\epsilon > 0$ [31]. Using ϵ, we can control



(a) Kariz Acceptance Ratio (b) MIP Acceptance Ratio (c) Comparison of Acceptance Ratio

Fig. 5. Acceptance ratio.

the trade-off between the accuracy and speed of our solution. Let (X^*, Y^*, Z^*) be the optimal solution. The number of actions performed will be at most $\frac{5|N|}{\epsilon} \ln \frac{B(X)+H(Y)}{B(X^*)+H(Y^*)}$ because the optimal solution is the lower bound for our solution. Since $\ln\big(B(X)+H(Y)\big)$ is polynomial in the size of the network and chain, the number of actions performed is also polynomial.

C. Update Layers

As the last piece of the puzzle, procedure *update-layers*() updates nodes in layers as shown in Alg. 3. From a layer $L(\overline{u})$ that traffic has already reached, every node $m \in L(\overline{u})$ is eliminated if this node does not allocate throughput of type $\overline{u}$ (lines 3-4). From other layers, nodes whose resources are allocated and hereafter cannot host corresponding VNF-instances are excluded (lines 5-7). Layers $L(\overline{s})$ and $L(\overline{t})$ are not updated.

Algorithm 3 Procedure *update-layers*()

1: **procedure** *update-layers*()
2: **for** $\overline{u} \in \overline{V}$ **do**
3: **if** traffic has reached $L(\overline{u})$ **then**
4: $L(\overline{u}) \leftarrow \{m | m \in L(\overline{u}),\ z_{m\overline{u}} > 0\}$
5: **else**
6: $L(\overline{u}) \leftarrow \{m | m \in L(\overline{u}),\ \exists v \in V_{\overline{u}}, \forall r : c_{mr} \geq d_{rv}\}$
7: **end if**
8: **end for**
9: **end procedure**

Through §V-A to V-B, we show that the running times of all *route*(.), *vnf-instances*(.), *improve*(.), and *update-layers*(.) are polynomial in the size of the network and chain. Hence, Kariz terminates in a polynomial time. Appendix B analyzes the time complexity of our algorithm.

VI. Evaluation

A. Experimental Setup

1) Simulated Network: The 6-ary Fat-tree, a common datacenter topology, is used as the simulated network, and contains 99 nodes (54 hosts and 45 switches) and 162 links providing full bi-sectional bandwidth. Hosts are equipped with a 20-core CPU and 2 Gbps network-adapter. The link capacities are 2 Gbps. This network is the largest network that we could run the implementation of DSFC model, as explained in § VI-A5, in a manageable time. The relative importance of allocating 1 Mbps of bandwidth over one link vs. one core CPU is 1% (i.e., $\frac{\text{number of CPU cores of a host}}{\text{bandwidth capacity of a host}}$).

TABLE III
Off-the-Shelf VNFs

Function	VNF	Throughput	CPU demand
firewall [1]	Level 1	100 Mbps	1 core
	Level 5	200 Mbps	2 core
	Level 10	400 Mbps	4 core
IDS	Bro [2]	80 Mbps	1 core
IPSec [3]	VSR1001	268 Mbps	1 core
	VSR1004	580 Mbps	4 core
WAN-opt. [6]	CCX770M	10 Mbps	2 core
	CCX1555M	50 Mbps	4 core

2) VNFs: We select the firewall, IDS, IPsec and WAN-opt. as functions. Table III reveals the VNFs used in the simulation. Since the CPU is the most restricted host resource and dominates the cost, we ignore memory and storage requirements.

3) Chains: Sources and targets are uniformly distributed in the network. Poisson distribution with the average of 1-chain per 100-seconds simulates the arrival rate. Chain lifetimes follow exponential distribution with an average of 3 hours.

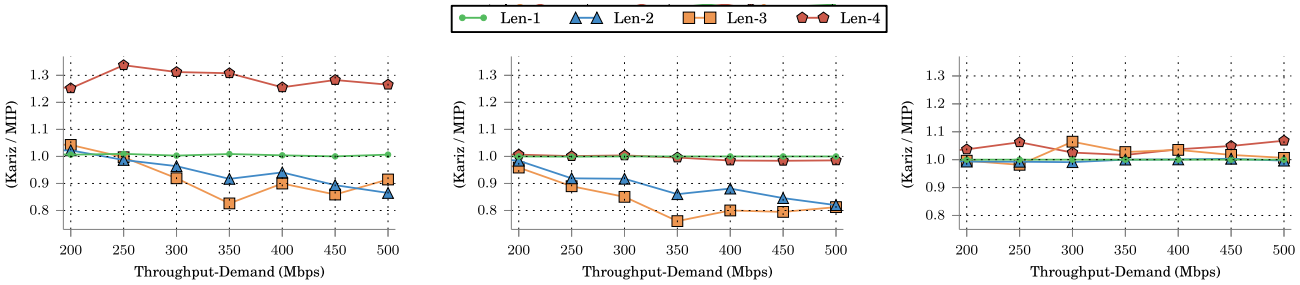
4) Parameters: We asses Kariz in respect to *throughput-demand* and *length* of chains. In each experiment, the throughput-demand is fixed to one of {200,250,300,…,500} Mbps, and one of the following chains is selected. Note that Len-i contains all functions of Len-i-1.

- Len-1: {firewall},
- Len-2: {firewall → IDS},
- Len-3: {firewall → IDS → IPSec}, and
- Len-4: {firewall → IDS → IPSec → WAN-opt.}

5) Evaluation Method: We compare Kariz with the model in § IV-B reffered as *MIP*. We implemented MIP using CPLEX. Note that MIP optimally deploys a single chain. Moreover, the tuning parameter of Kariz is set to $\epsilon = 32$. Thus, an action is performed if it improves the current solution by $\sim 6\%$. With fixed parameters, we repeat each experiment 10 times for every 1000 chains generated, and report the average.

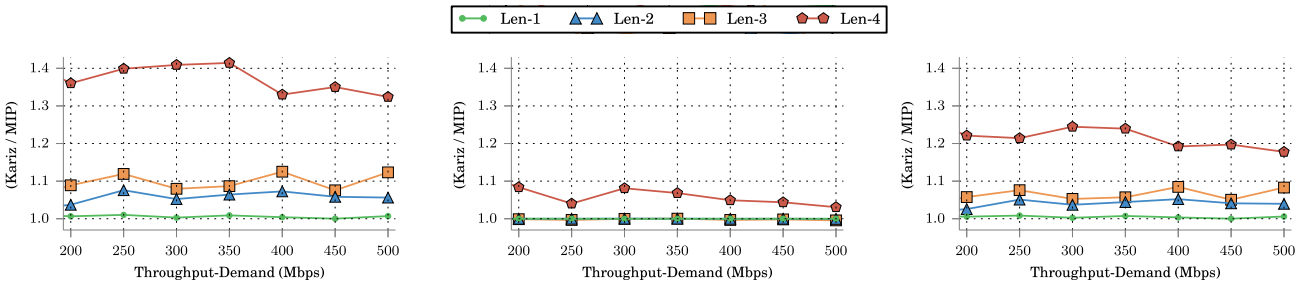
B. Acceptance Ratio

Fig. 5a and Fig. 5b depict the acceptance ratios of Kariz and MIP, respectively. The values are the average acceptance ratios from 10 experiments. As expected, the longer chains with higher throughput-demand have less chance to be accepted.



(a) Bandwidth Utilization (b) CPU Utilizaiton (c) VNF Utilization

Fig. 6. Comparison of resource utilization.



(a) Bandwidth Allocation Cost (b) CPU Resource Allocation Cost (c) Total Cost

Fig. 7. Operational costs.

The low acceptance ratio for Len-4 is due to the resource hunger of these chains, especially for WAN-opt. VNFs.

The range of numbers of chains accepted by Kariz vs. MIP in Fig. 5c are: 100-100% for Len-1, 82-99% for Len-2, 76-96% for Len-3, and 89-97% for Len-4. Considering the chain length and throughput-demand impacts in Fig. 5c, Kariz performs closely to MIP. It might be expected that increasing the length of chain and throughput-demand should cause Kariz to have a lower acceptance ratio than MIP. However, Kariz has better results for Len-4 than Len-3 and Len-2, especially for 500 Mbps throughput-demand. Recall from § V-B that Kariz attempts to improve the solution after deployment of every function of a chain. Since, Len-4 includes all functions of Len-3 and Len-2 chains (see § VI-A4), the expense of more improvement rounds increases the chance of adjusting the earlier solution. All in all, Kariz has a competitive acceptance ratio, within 76-100% of MIP.

C. Resource Utilization

Fig. 6 compares the resource utilization of Kariz with MIP's. Bandwidth/CPU utilization for Kariz and MIP are the ratio of allocated bandwidth/CPU resources over aggregated bandwidth/CPU capacities in the network. For VNF resources, the reports are the average of per-function throughput utilization.

The bandwidth utilization ratios as depicted in Fig. 6a are 100-101% for Len-1, 86-102% for Len-2, 83-104% for Len-3, and 125-134% for Len-4. Fig. 6a and Fig. 5c show that Kariz efficiently utilizes the bandwidth resources for Len-1, Len-2, and Len-3 for various throughput-demands. Regarding Len-4, Kariz's efficiency in utilizing bandwidth resources decreases.

The CPU utilization ratios are in the range of 100-100% for Len-1, 82-98% for Len-2, 76-96% for Len-3, and 98-101% for Len-4, as observed in Fig. 6b. According to Fig. 6b and Fig. 5c, Kariz utilizes CPUs efficiently, close to MIP's.

Finally, the VNF utilization ratios vs. MIP are shown in Fig. 6c. The following ranges are reported: 100-100% for Len-1, 99-100% for Len-2, 98-106% for Len-3, and 102-107%. Kariz utilizes VNF instances with an efficiency close to that of MIP for different lengths and throughput demands.

D. Operational Costs

Fig. 7 shows Kariz's costs vs MIP's. We collect Kariz's and MIP's average of per-chain costs. The reported values are the ratio of Kariz's and MIP's costs.

As shown in Fig. 7a on average, Kariz allocates bandwidth resource vs. MIP in the range 100-101% for Len-1, 104-108% for Len-2, 108-113% for Len-3, and 132-141% for Len-4. Regarding CPU as presented in Fig. 7b, on average, the same number of CPU cores is allocated for Len-1, Len-2, and Len-3. For Len-4, Kariz allocates 3-8% more CPU cores.

Finally, in respect to the total operational cost in Fig. 7c, the following cost ratios vs MIP are observed: 100-101% for Len-1, 103-105% for Len-2, 105-108% for Len-3, and 117-124% for Len-4. Kariz is more cautious to allocate CPUs than to allocate bandwidth showing that improvement rounds (see § V-B) optimize the total cost by releasing CPUs while allocating more bandwidth. In summary,

Kariz incurs a competitive per-chain cost that is less than 124% of MIP's.

VII. Conclusion and Future Work

The limitations of current optimization models restrict a chain's throughput to resources of a physical-machine and result in sub-optimal resource utilization and service performance. In this paper, we presented distributed service function chaining (DSFC) to overcome these limitations. DSFC decouples a chain's throughput from physical-machines by placing VNF-instances of a function in multiple machines. Further, DSFC optimizes network utilization by coordinating the deployment operations. We formulated DSFC using a mixed integer programming (MIP) model and proved its NP-Hardness. For larger scales, we proposed and evaluated a heuristic called Kariz. The experimental results for various chain lengths and throughput demands demonstrate that Kariz achieves competitive cost and acceptance ratio compared to the MIP model. In future, we plan to analyze the lower bound of approximation algorithms for DSFC problem. We aim to devise an approximation algorithm close to the lower bound.

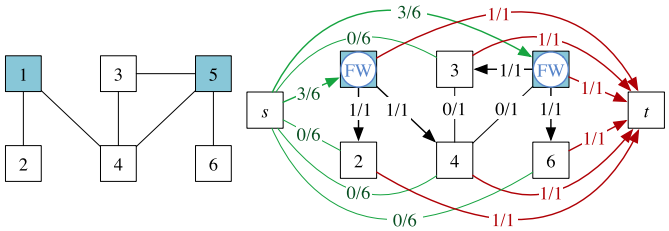
Appendix A
NP-Hardness Proof

In this section, we prove that the Distributed Service Function Chaining (DSFC) problem is NP-Hard. We use a *reduction* from the NP-Hard *minimum dominating set* problem. A dominating set of a graph is a set of nodes so that each node is either a member or adjacent to at least a member of this set. The goal is to find a dominating set with minimum size.

Given a graph $\boldsymbol{G}$ with $\boldsymbol{n}$ nodes as an instance of the dominating set problem, we create an instance of DSFC and prove that there is a dominating set of size $\boldsymbol{k}$ if and only if there is a solution of cost $3\boldsymbol{n}+\boldsymbol{k}$ in the DSFC instance.

Recall from § IV-A, an instance of DSFC is defined with resources, a network, a service chain, and VNFs. In our reduced instance, CPU is the only host resource. We define the network to be the same as $\boldsymbol{G}$ with two extra nodes s and t. These two nodes are connected to all nodes of $\boldsymbol{G}$. Nodes s and t have 1 CPU-core, and others have 2 CPU-cores. Incident links on s have the capacity $\boldsymbol{n}$, and the capacity of other links is 1. The chain is defined as $\overline{s} \rightarrow$ FW $\rightarrow \overline{t}$. Endpoints $\overline{s}$ and $\overline{t}$ respectively correspond to source s and target t in the network, and FW is a firewall SF. Moreover, there is a single firewall VNF demanding 2 CPU-cores and providing throughput capacity $\boldsymbol{n}$. On this instance of DSFC, the goal is to install firewall VNFs to process a *flow* of size $\boldsymbol{n}$ from s to t. Note that these VNFs require 2 CPU-cores and cannot be placed in s or t. Here, by 'flow', we mean the traffic that is to be sent from s to t. We define α_r ($r = \text{CPU}$) and β to be 1, hence the total cost of DSFC is the total number of allocated CPU-cores (resource cost) and bandwidth. Fig. 8 depicts this reduction. Clearly, the DSFC instance can be constructed in polynomial time from the dominating set instance.

To prove the hardness, we show that there is a dominating set of size $\boldsymbol{k}$ if and only if there is a solution for the DSFC instance with cost $3\boldsymbol{n}+\boldsymbol{k}$. We start with the easy direction:



(a) Dominating Set (b) DSFC

Fig. 8. To send a flow of size $\boldsymbol{n}=6$ from s to t, a flow of size 3 is sent to nodes 1 and 5. These nodes then send a flow of size 1 to any node that they dominate. Firewalls are placed at dominating nodes 1 and 5.

Lemma 1: If there is a solution of size $\boldsymbol{k}$ for the dominating set problem, then there is a solution of cost $3\boldsymbol{n}+\boldsymbol{k}$ for the DSFC problem.

Proof: Let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ denote the nodes in the dominating set, and $\boldsymbol{a_i}$ ($1 \leq \boldsymbol{i} \leq \boldsymbol{k}$) denote the number of nodes $\boldsymbol{v_i}$ dominates. If a node is dominated by more than one node, we count it only once (arbitrary assign it to one node in the dominating set). Note that we have $\sum_{\boldsymbol{i}} \boldsymbol{a_i} = \boldsymbol{n}-\boldsymbol{k}$.

In the DSFC instance, we send a flow of size $\boldsymbol{a_i}+1$ from s to any $\boldsymbol{v_i}$. Doing so results in bandwidth cost of $\boldsymbol{n}$. We also send a flow of size 1 from $\boldsymbol{v_i}$ to any of the nodes that it dominates; this requires a bandwidth of $\boldsymbol{a_i}$ for $\boldsymbol{v_i}$ and in total bandwidth of $\boldsymbol{n}-\boldsymbol{k}$ for all dominating nodes. Finally, we send a flow of size 1 from all nodes (except s) to t. This results in bandwidth cost of $\boldsymbol{n}$ (see Fig. 8b). We install a FW for the service chain in each node in the dominating set, resulting in resource cost of $2\boldsymbol{k}$. In total, the cost is $\boldsymbol{n}+(\boldsymbol{n}-\boldsymbol{k})+\boldsymbol{n}+2\boldsymbol{k}=3\boldsymbol{n}+\boldsymbol{k}$. □

To prove the other side of the reduction, we start with Lemma 2.

Lemma 2: Given a solution of the DSFC problem with cost $\boldsymbol{c}$, one can achieve a solution of cost no more than $\boldsymbol{c}$ in polynomial time, for which the following properties for each node other than s, t hold: (1) the total inflow received through nodes other than s is at most 1; (2) the inflow is through s or a node that receives inflow through s; moreover, (3) FWs are placed at nodes receiving some flow directly from s; (4) each node receives either all or none of its inflow from s.

Proof: We modify the solution to satisfy properties (1)-(4) in the same order without affecting previously satisfied properties. In this process, the cost of the solution is never increased. To satisfy (1), assume there is node $\boldsymbol{u}$ with inflow $\boldsymbol{x} > 1$ from node(s) other than s. This assumption implies a bandwidth cost of at least $2\boldsymbol{x}$ for the flow passing at least another node between s and $\boldsymbol{u}$. In the new solution, we remove this flow and send a flow of size $\boldsymbol{x}$ from s to $\boldsymbol{u}$ and place a FW at $\boldsymbol{u}$. Doing so gives a bandwidth cost of $\boldsymbol{x}$ and resource cost of 2. The increase in cost is no more than $\boldsymbol{x}+2-2\boldsymbol{x} \leq 0$. Property (2) follows directly from (1). To satisfy (3), note that by (1), the flow between nodes excluding s and t form a forest. Placing FWs only in the roots in the forest does not increase the cost. For (4), assume a node receives an inflow of $\boldsymbol{x}$ through s and an inflow of 1 through another node. By (3), there is a FW at $\boldsymbol{u}$. In the new solution, we send a flow of $\boldsymbol{x}+1$ from s to $\boldsymbol{u}$ and remove the flow from the other node. □

We use Lemma 2 to prove the other side of the reduction:

Lemma 3: If there is a solution of cost $3\boldsymbol{n}+\boldsymbol{k}$ *for the DSFC problem, then there is a solution of size* $\boldsymbol{k}$ *for the dominating set problem.*

Proof: First, we apply Lemma 2 to achieve a solution with the desired properties. We refer to the nodes that receive flow through s as *critical nodes*. By property (4), a node receive all or none of its inflow from s. By (3), there is a FW located in critical nodes. By (1), each non-critical node has inflow of 1 and by (2), such node receives this inflow through a critical node. In other words, the graph formed by flows (excluding t), is a tree of diameter 2 rooted at s. Let $\boldsymbol{m}$ denote the number of critical nodes; the resource cost for FWs would be $2 \times \boldsymbol{m}$. The bandwidth cost is $3\boldsymbol{n}-\boldsymbol{m}$: a cost of $\boldsymbol{n}$ for the outflow of s, another cost of $\boldsymbol{n}$ for the inflow of t, and an extra bandwidth cost of $\boldsymbol{n}-\boldsymbol{m}$ for the flow from critical nodes to non-critical ones (recall that the inflow of each non-critical node is 1). In conclusion, the total cost of the solution is $3\boldsymbol{n}+\boldsymbol{m}$, i.e., we have $3\boldsymbol{n}+\boldsymbol{m}=3\boldsymbol{n}+\boldsymbol{k}$. In other words, the number of critical nodes is $\boldsymbol{k}$. On the other hand, by (1) and (2) each non-critical node has an inflow of exactly 1 through a critical node. Hence, critical nodes form a dominating set of size $\boldsymbol{k}$. □

From Lemmas 1 and 3, Theorem 1 is direct.

Theorem 1: Finding the solution with minimum cost for DSFC is NP-hard.

APPENDIX B
TIME COMPLEXITY ANALYSIS

Kariz routes traffic and installs VNF instances layer by layer. For each layer, Kariz (i) finds a feasible initial solution, (ii) improves this solution, and (iii) updates layers accordingly. One can verify that the second step dominates the time complexity for the computation performed in each layer. In this step, Kariz performs repeatedly either best $add(.)$ or best $open(.)$. The number of performed actions depends on the quality of the initial solution and is at most $\frac{5|N|}{\epsilon}\ln\frac{B(X)+H(Y)}{B(X^*)+H(Y^*)}$. At the worst case, the initial solution is $O(|N|)$ worse than the optimal solution (placing an 'almost' idle VNF in each subtrate node). Thus, the number of performed actions is in $O(|N|\log|N|)$. Finding the best $add(.)$ action is examining at most $\overline{b}|N||\overline{V}|$ actions each of which entails MCFP problem. Let $\Psi(G)$ be time complexity of solving an instance of MCFP problem [11]. The complexity of finding the best $add(.)$ action is $\overline{b}|N||\overline{V}|\Psi(G)$. Each $open(.)$ action also involves solving MCFP problem. For the best $open(., M, L(\overline{v}), .)$, Kariz finds a subset M in $L(\overline{u})$ that at worst is in $O(|N|^2)$. Thus the complexity of finding the best $open(.)$ action is in $O(|N|^2)\Psi(G)$. In total, the time complexity of running the second step for each layer is $O(|N|^2\log(|N|))\Psi(G)$. All in all, since the second step is performed for $|\overline{V}|$ layers, Kariz's time complexity is in $O(|\overline{V}||N|^3\log|N|\Psi(G))$. We note that this complexity does not reflect the typical time complexity of the algorithm, and in most cases Kariz runs much faster than this upper bound.

REFERENCES

[1] *Barracuda WAF*. Accessed: Mar. 20, 2017. [Online]. Available: https://goo.gl/QmLv8E

[2] *Bro*. Accessed: Mar. 20, 2017. [Online]. Available: https://www.bro.org/sphinx/cluster/index.html

[3] *HP Virtual Router Series*. Accessed: Mar. 20, 2017. [Online]. Available: http://goo.gl/hPWrTt

[4] *OpenFlow Switch Specification v.1.3.1*. Accessed: Mar. 20, 2017. [Online]. Available: https://goo.gl/llwvyE

[5] *Service Function Chaining Use-Cases*. Accessed: Mar. 20, 2017. [Online]. Available: https://goo.gl/fgRbCp

[6] *SteelHead Product Family Spec Sheet*. Accessed: Mar. 20, 2017. [Online]. Available: http://goo.gl/g2XfNs

[7] M. Alizadeh *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. SIGCOMM*, 2014, pp. 503–514.

[8] B. Anwer, T. Benson, N. Feamster, and D. Levin, "Programming slick network functions," in *Proc. SOSR*, 2015, Art. no. 14.

[9] K. Argyraki *et al.*, "Can software routers scale?" in *Proc. PRESTO*, 2008, pp. 21–26.

[10] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *Proc. CNSM*, 2015, pp. 50–56.

[11] R. Becker, M. Fickert, and A. Karrenbauer, "A novel dual ascent algorithm for solving the min-cost flow problem," in *Proc. ALENEX*, 2016, pp. 151–159.

[12] P. C. Chu and J. E. Beasley, "A genetic algorithm for the multidimensional knapsack problem," *J. Heuristics*, vol. 4, no. 1, pp. 63–86, 1998.

[13] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer, "Predicting the resource consumption of network intrusion detection systems," in *Proc. RAID*, 2008, pp. 135–154.

[14] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multi-commodity flow problems," in *Proc. SFCS*, 1975, pp. 184–193.

[15] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *Proc. USENIX NSDI*, 2014, pp. 533–546.

[16] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella, "Toward software-defined middlebox networking," in *Proc. HotNets*, 2012, pp. 7–12.

[17] A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella, "Stratos: Virtual middleboxes as first-class entities," UW-Madison, Madison, WI, USA, Tech. Rep. TR1771, 2012.

[18] A. Gember-Jacobson *et al.*, "OpenNF: Enabling innovation in network function control," in *Proc. SIGCOMM*, 2014, pp. 163–174.

[19] C. Ghribi, M. Mechtri, and D. Zeghlache, "A dynamic programming algorithm for joint VNF placement and chaining," in *Proc. CAN*, 2016, pp. 19–24.

[20] A. V. Goldberg and R. E. Tarjan, "Finding minimum-cost circulations by canceling negative cycles," *J. ACM*, vol. 36, no. 4, pp. 873–886, 1989.

[21] S. Guha, A. Meyerson, and K. Munagala, "Hierarchical placement and network design problems," in *Proc. FOCS*, 2000, pp. 603–612.

[22] A. Hirwe and K. Kataoka, "LightChain: A lightweight optimisation of VNF placement for service chaining in NFV," in *Proc. NetSoft*, Jun. 2016, pp. 33–37.

[23] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?" in *Proc. IMC*, 2011, pp. 181–194.

[24] D. Joseph and I. Stoica, "Modeling middleboxes," *IEEE Netw.*, vol. 22, no. 5, pp. 20–25, Sep./Oct. 2008.

[25] K. Argyraki *et al.*, "Understanding the packet forwarding capability of general-purpose processors," Intel Res., Berkeley, CA, USA, Tech. Rep. IRB-TR-08-44, 2008.

[26] R. Kapoor, A. C. Snoeren, G. M. Voelker, and G. Porter, "Bullet trains: A study of NIC burst behavior at microsecond timescales," in *Proc. CoNEXT*, 2013, pp. 133–138.

[27] T.-W. Kuo, B.-H. Liou, K. C.-J. Lin, and M.-J. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," in *Proc. INFOCOM*, 2016, pp. 1–9.

[28] H. Long, Y. Shen, M. Guo, and F. Tang, "LABERIO: Dynamic load-balanced routing in OpenFlow-enabled networks," in *Proc. IEEE AINA*, Mar. 2013, pp. 290–297.

[29] T. Lukovszki, M. Rost, and S. Schmid, "It's a match!: Near-optimal and incremental middlebox deployment," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 1, pp. 30–36, 2016.

[30] M. C. Luizelli *et al.*, "A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining," *Comput. Commun.*, vol. 102, pp. 67–77, Apr. 2016.

[31] M. Mahdian and M. Pál, "Universal facility location," in *Proc. ESA*, 2003, pp. 409–421.

[32] H. Moens and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *Proc. IEEE CNSM*, Nov. 2014, pp. 418–423.

[33] M. Pál, T. Tardos, and T. Wexler, "Facility location with nonuniform hard capacities," in *Proc. FOCS*, 2001, pp. 329–338.

[34] S. Palkar, "E2: A framework for NFV applications," in *Proc. SOSP*, 2015, pp. 121–136.
[35] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Comput. Netw.*, vol. 31, pp. 2435–2463, Dec. 1999.
[36] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying middlebox policy enforcement using SDN," in *Proc. SIGCOMM*, 2013, pp. 27–38.
[37] P. Quinn and T. Nadeau, "Service function chaining problem statement," IETF, Fremont, CA, USA, Tech. Rep. RFC 7498, 2014.
[38] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/Merge: System support for elastic execution in virtual middleboxes," in *Proc. NSDI*, 2013, pp. 227–240.
[39] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proc. USENIX NSDI*, 2012, p. 24.
[40] S. Sinha, S. Kandula, and D. Katabi, "Harnessing TCP's burstiness with flowlet switching," in *Proc. HotNets*, 2004, pp. 1–6.
[41] R. Sommer, M. Vallentin, L. De Carli, and V. Paxson, "HILTI: An abstract execution environment for deep, stateful network traffic analysis," in *Proc. IMC*, 2014, pp. 461–474.
[42] J. A. Tomlin, "Minimum-cost multicommodity network flows," *Oper. Res.*, vol. 14, no. 1, pp. 45–51, 1966.
[43] J. Verdú, M. Nemirovsky, and M. Valero, "MultiLayer processing—An execution model for parallel stateful packet processing," in *Proc. ANCS*, 2008, pp. 79–88.
[44] L. Wang, Z. Lu, X. Wen, R. Knopp, and R. Gupta, "Joint optimization of service function chaining and resource allocation in network function virtualization," *IEEE Access*, vol. 4, pp. 8084–8094, 2016.
[45] T. Wen, H. Yu, G. Sun, and L. Liu, "Network function consolidation in service function chaining orchestration," in *Proc. ICC*, 2016, pp. 1–6.
[46] Y. Zhang *et al.*, "StEERING: A software-defined networking for inline service chaining," in *Proc. ICNP*, 2013, pp. 1–10.
[47] W. Zh *et al.*, "An untold story of middleboxes in cellular networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 374–385, Aug. 2011.

Milad Ghaznavi is currently pursuing the Ph.D. degree with the School of Computer Science, University of Waterloo. His research interest includes distributed systems, network and systems, and algorithms. He was a recipient of the David R. Cheriton Graduate Scholarship at the University of Waterloo.

Nashid Shahriar is currently pursuing the Ph.D. degree with the School of Computer Science, University of Waterloo. His research interest includes future network architectures, network virtualization, and network function virtualization. He was a recipient of David R. Cheriton Graduate Scholarship at the University of Waterloo.

Shahin Kamali was a Post-Doctoral Fellow with the Massachusetts Institute of Technology. He is currently an Assistant Professor with the Department of Computer Science, University of Manitoba. His research broadly covers design, analysis, applications, and limitations of approximation and online algorithms, the applications of online algorithms in text compression, graph partitioning, and resource allocation in cloud.

Reaz Ahmed received the Ph.D. degree in computer science from the University of Waterloo in 2007. His research interests include future Internet architectures, information-centric networks, network virtualization, and content sharing peer-to-peer networks with focus on search flexibility, efficiency, and robustness. He received the IEEE Fred W. Ellersick Award in 2008.

Raouf Boutaba (F'12) received the M.Sc. and Ph.D. degrees in computer science from the University Pierre and Marie Curie, Paris, in 1990 and 1994, respectively. He is currently a Professor of computer science with the University of Waterloo, Canada. His research interests include resource and service management in networks and distributed systems. He is a fellow of the Engineering Institute of Canada and the Canadian Academy of Engineering.