Breaking Service Function Chains with Khaleesi

Sara Ayoubi, Shihabur Rahman Chowdhury, and Raouf Boutaba
David R. Cheriton School of Computer Science, University of Waterloo
{sayoubi | sr2chowdhury | rboutaba}@uwaterloo.ca

***Abstract*—Network Function Virtualization (NFV) has recently emerged as a means to replace vendor specific, purpose built equipment with commodity hardware and leverage the open APIs and application orchestration for on demand deployment and scaling of network services. A well studied problem in NFV is the orchestration of Service Function Chains, (SFCs), *i.e.*, a set of Virtual Network Functions (VNFs) chained together to realize a network service. State-of-the-art literature on SFC orchestration assumes a strict traversal order of VNFs in an SFC and less attention has been paid to SFCs with relaxed VNF orderings. In this paper, we address the problem of *Flexible Service Function Chain Orchestration* that jointly allocates compute and network resources for SFCs while considering a relaxed traversal order for some pairs of VNFs. We propose *Khaleesi*, a suite of solutions that consists of: (i) an Integer Linear Program (ILP) for optimally solving the problem; and (ii) a heuristic algorithm to scale to larger instances of the problem. Our simulation results show that flexible SFCs can increase revenue earned per unit cost by as much as ≈10% compared to a rigid SFC.**

I. Introduction

Network Function Virtualization (NFV) [1] has received significant traction in recent years for its ability to decouple packet processing software, *i.e.*, *Network Functions* (NFs) from specialized hardware *middleboxes*. NFV proposes to run the NFs as *Virtual Network Functions (VNFs)* on commodity hardware and leverages advances in application orchestration for on-demand provisioning of Service Function Chains (SFCs), *i.e.*, an ordered sequence of VNFs that implements a network service. The capability of on-demand service provisioning and the consolidation of multiple NFs on commodity hardware enable the network operators to reduce their operational and capital expenditures.

However, the full benefits offered by NFV cannot be fully reaped without efficient resource allocation mechanisms for provisioning the SFCs. As a result, a significant body of research has been dedicated to address the problem of SFC orchestration, *i.e.*, joint allocation of compute and network resources to provision SFCs with Quality of Service (QoS) (*e.g.*, minimum bandwidth or maximum delay) requirements. SFC orchestration has been shown to be NP-hard [2] and a number of its variants have been studied in the research literature [3]. Despite the significant research efforts in solving SFC orchestration, little attention has been paid to the actual semantics of the VNFs while allocating resources for them. As a result, SFC has been mostly considered as a rigid sequence of VNFs, *i.e.*, the order of the VNFs cannot be modified. In this work, we take a closer look at the semantics of VNFs and show that VNF traversal order can be changed without modifying the semantics of an SFC, and that such flexibility can be leveraged to perform better resource allocation for SFCs.

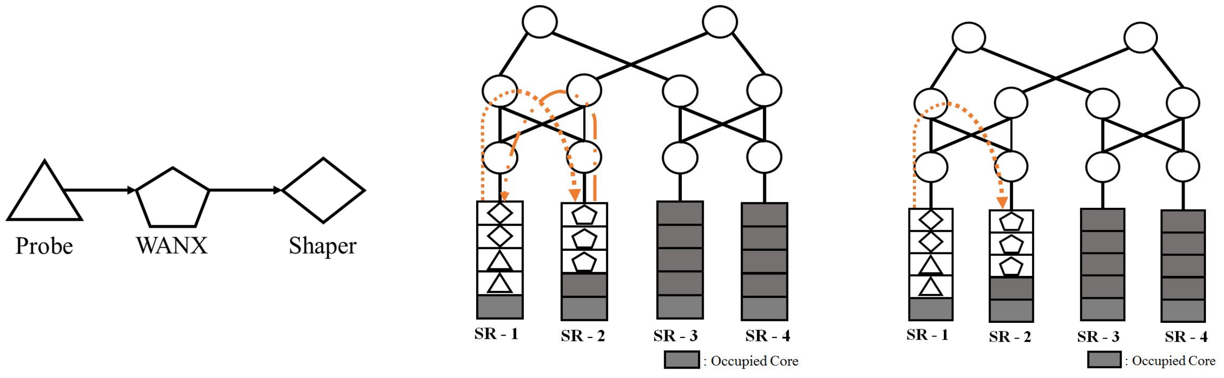
As an illustrative example, consider the following VNFs: a WAN optimizer that compresses and decompresses HTTP payload and a Probe that counts flows with a given layer 3 and layer 4 signature. These VNFs work on disjoint parts of a packet. Therefore, if they are next to each other in an SFC, swapping their order will neither affect the set of packets exiting the chain, nor the internal state of the VNFs (we call such VNFs *reorder-compatible*). Now consider the example SFC in Fig. 1(a), where the Probe, WAN Optimizer and Shaper requires 2, 3, and 2 CPU cores, respectively. If we consider the SFC to be rigid (*i.e.*, the order of VNFs cannot be modified) then the only possible provisioning solution is the one shown in Fig. 1(b). However, if we consider the reorder-compatibility of Probe and the WAN Optimizer, then we can swap their order in the SFC and provision the SFC as shown in Fig. 1(c). Note that, by leveraging the flexibility in VNF ordering, we provisioned the same SFC with 50% less network bandwidth.

Considerations for flexible SFCs are not entirely new. Early works in this area [4]–[6] proposed languages and data models to represent flexible SFC requests. However they do not discuss how the flexibility can be determined in the first place. More recently, Parabox [7] and NFP [8] proposed to parallelize VNF execution in an SFC by introducing additional components. In contrast, we do not assume any additional components for changing the order of VNFs in an SFC. Moreover, no quantifiable results exist in the research literature that demonstrates if any benefit can be gained from flexibility in VNF ordering. In this paper, we fill this gap in research literature with the following contributions:

- Theoretical foundation for determining reorder compatibility of VNFs and the mathematical models to represent such flexibility in an SFC.
- The first quantifiable result showing the benefits of flexible SFC orchestration over its rigid counterpart. Our empirical results demonstrate as much as 10% improvement in revenue earned per unit cost compared to rigid SFCs.
- *Khaleesi*[1], a suite of solutions to the Flexible SFC orchestration problem consisting of: (i) *OPT-Khaleesi*, an Integer Linear Program (ILP) formulation for optimally solving the flexible SFC orchestration problem. To the best of our knowledge, this is the first optimal solution proposed for such problem, and (ii) *FAST-Khaleesi*, A

[1]A character in popular fantasy novel *"A Song of Ice and Fire"*, who is also known as *the breaker of chains*

ISBN 978-3-903176-08-9 2018 IFIP



(a) Service Function Chain (b) Placement of a Rigid SFC (c) Flexible SFC (Probe and WANX swapped)

Fig. 1. Motivational Example

heuristic algorithm to solve larger instances of the problem. Simulation results show that *FAST-Khaleesi* allocates $\approx 2\times$ extra resources and accepts $\approx 20\%$ less SFC requests on average compared to the optimal solution.

The rest of the paper is organized as follows. We begin with a discussion of related work in Section II. Next, we present the theoretical foundations for identifying re-order compatible VNFs in Section III. In Section IV we present the system model and formally define the problem. We present our ILP formulation in Section V, followed by the heuristic in Section VI. Our evaluation results are presented in Section VII. In Section VIII we provide a brief discussion on the operational aspects that may limit the flexibility of SFCs. Finally, we conclude with some future research directions in Section IX.

II. Related Work

Since the publication of introductory white paper in late 2013, research in NFV has gained significant traction over the past few years. In the following, we discuss the state-of-the-art in SFC orchestration with a specific focus on research that considers relaxed order of VNFs in an SFC.

SFC orchestration is one of the most well studied problem in NFV. It has been addressed with different objective functions *e.g.*, maximizing the number of admitted SFCs [9], minimizing operational cost [2], minimizing the number of servers used [10], minimizing network resource utilization [11], minimizing the number of VNF instances used [4], [12] among others. For a comprehensive survey on resource allocation in NFV readers are referred to [3]. Approaches to solve these problems include variants of Linear Programming [4], [10], [11], Cut-and-solve method [9], polynomial time heuristic design [2], [12], approximation algorithm [13], *etc*. However, majority of these works assume a rigid SFC and do not leverage the flexibility of reorder compatibility between NFs.

Only a handful of research has considered relaxed ordering of NFs in an SFC. For instance [5] proposes a context free grammar to represent an SFC request with flexible parts. The flexible parts are segments of NFs that can be traversed in any arbitrary order. An extension of this work is presented in [6] where the authors propose a YANG data model to represent flexible structures. The authors also propose a Pareto-optimal solution and a heuristic to allocate resources for such SFCs with flexible parts. However, works such as [5], [6], [14] neither quantify the advantages in resource allocation stemming from flexible VNF ordering in an SFC over its rigid counterpart, nor do they shed light on the aspects that may affect such flexibility. More recently, Parabox [7] and NFP [8] proposed to relax the strict ordering of VNFs in an SFC and parallelize some of them to reduce end-to-end latency. However, to do so, additional components were introduced for splitting incoming packets to parallelized functions and also to combine their output. This incurs overhead in terms of network resources and processing delays. In contrast, we identify “re-order compatible” VNFs that can be swapped without adding any additional functions.

III. Re-order Compatibility of VNFs

Flexible SFCs can bring benefits in terms of resource allocation, thereby freeing up resources for more SFCs to be admitted. This can indeed increase infrastructure provider’s revenue in the long run. Flexible SFCs have been preliminarily addressed before in [6] and [7]. However, in [6], the flexibility of the chain is an input to the orchestration problem. In [7], [8], the authors identify *independent* VNFs to increase parallelism in the chain. The concept of independent VNFs is somewhat similar to reorder compatible VNFs, however, not every independent pair of VNFs is re-order compatible. To the best of our knowledge, no existing work has yet formally defined re-order compatible VNFs. In the following, we lay the necessary theoretical foundation for identifying reorder compatible VNFs.

In a nutshell, two VNFs are considered re-order compatible if swapping their order in an SFC does not violate the SFC’s semantics, *i.e.*, results in two *semantically equivalent* SFCs. We formally define semantically equivalent SFCs as follows:

Definition 1. ***Semantically Equivalent SFC:*** *Two SFCs S_1 and S_2 composed of the same set of VNFs, $\mathcal{F}$, in different order are semantically equivalent if: (i) for an ordered sequence of input packets p_{in}, both S_1 and S_2 produce identical ordered sequence of output packets, and (ii) after processing a packet*

$p \in p_{in}$ the internal state of any VNF $f_i \in \mathcal{F}$ is identical in both S_1 and S_2.

As a packet traverses an SFC, a VNF in the SFC performs any combination of the following three actions: (i) reads from the packet, (ii) modifies the packet, and (iii) updates its own internal state. For instance, while traversing a NAT, the source IP and source MAC address of a packet are modified, as well as the NAT's address translation table. In another instance, a probe may keep a count of the number of UDP packets that are sent/received and updates the count after a packet passes through it. Since SFCs provide a form of value-added service, therefore it is important to ensure that any flexible structure of the SFC provides the same service to the flows as well as the constituent VNFs have identical internal states.

To formalize this, we refer to the set of packet[2] fields that a VNF reads or modifies as "interest fields", denoted as $\mathcal{H}_f^i$, and the set of packet fields that affects the internal state of a VNF as "state fields", denoted as $\mathcal{H}_f^s$. Every $h \in \mathcal{H} = \mathcal{H}_f^i \cup \mathcal{H}_f^s$ is expressed as a (byte_offset, byte_length) pair (*e.g.*, source MAC can be expressed as a pair (6, 6)), which allows us to express interest fields and header fields in a protocol agnostic way (similar to [15]). By comparing the interest and state fields of two VNFs, we can determine their re-order compatibility.

Two VNFs are re-order compatible when their interest and state fields are mutually exclusive (*e.g.*, an application-layer firewall and a network-layer firewall). Furthermore, even when two VNFs share the same interest and/or state fields, as long as they do not modify the shared interest and/or state fields their processing functions remain independent (*e.g.*, a probe and a Deep Packet Inspector (DPI)). For each VNF, we represent the set of interest and state fields by a $|\mathcal{H}| \times$ 3 binary matrix $\mathcal{M}$, illustrated in Table I. The rows in $\mathcal{M}$ represent the different fields $h \in \mathcal{H}$. The columns r and w indicate whether this particular VNF, f, reads and/or modifies h, respectively. The column x indicates whether h affects the internal state of f, denoted by x = if($h \in \mathcal{H}_f^s$); $x \in \{0,1\}$. Given two VNFs u and v and their corresponding matrices $\mathcal{M}_u$ and $\mathcal{M}_v$, respectively, u and v are re-order compatible if:

$$\begin{aligned} \forall h \in \mathcal{H}, \forall (k,k') \in \{(u,v),(v,u)\} : \\ (\mathcal{M}_k[h][r] \wedge \mathcal{M}_{k'}[h][w]) \vee \\ (\mathcal{M}_k[h][w] \wedge \mathcal{M}_{k'}[h][w]) \vee \\ (\mathcal{M}_k[h][x] \wedge \mathcal{M}_{k'}[h][r]) \;=\; 0 \end{aligned} \quad (1)$$

Finding the interest and state fields of a VNF is a non-trivial task and is in fact a separate problem on its own. Different approaches have been taken in the past including middlebox modeling [16] [17] [18], header space analysis [19], white-box testing [20], black-box testing [21], *etc.* However, it is an orthogonal problem and is out of the scope of this paper.

A. Illustrative Example

Table II illustrates a re-order compatibility matrix for some commonly deployed NFs [22], namely firewall, web proxy, Intrusion Prevention System (IPS), Traffic Shaper, NAT, payload-DPI, and WAN Optimizer (WANX). We obtained the interest and state fields for these different VNFs by investigating existing middlebox models [16], middlebox catalog [23], IETF drafts [24], Click configuration files [25], and related research literature [7]. By applying (1) on the obtained interest and state fields, we obtained the matrix in Table II.

We observe that a network Firewall (besides changing the MAC address) typically examines layer 2-4 headers, *e.g.*, source IP, destination IP, and port numbers, and either forwards or drops a packet. An IPS analyzes the packet (header and/or payload) and takes automated actions (drops packet, blocks traffic, sends alarm, or resets connection). Since a network firewall and an IPS perform "read-only" actions on common interest fields they can be swapped without affecting chain semantics. A traffic shaper classifies network traffic for QoS, a payload-DPI inspects the packet payload and raises an alarm if the packet matches a malicious signature. These VNFs are clearly re-order compatible with a network Firewall. Finally, a WANX can perform functions such as payload compression/de-compression, QoS tagging *etc.*, which do not affect the interest and state fields of a network firewall. This renders them also re-order compatible. However, a network firewall and a network probe may not be re-order compatible because of the Probe's internal state. For instance, if a probe is counting the number of incoming connections to port 80, placing a firewall before the probe will yield a different count than placing it after the probe.

TABLE I
INTEREST & STATE FIELD MATRIX $\mathcal{M}$

	r	w	x
h_1			
h_2			
....			
$h_{\lvert\mathcal{H}\rvert}$			

TABLE II
REORDER COMPATIBILITY MATRIX

	Firewall	Proxy	IPS	Shaper	NAT	DPI	WANX	Probe
Firewall			✓	✓		✓	✓	
Proxy			✓	✓		✓		
IPS	✓	✓		✓				✓
Shaper	✓	✓	✓			✓	✓	
NAT							✓	
DPI	✓	✓		✓				✓
WANX	✓			✓	✓			✓
Probe			✓			✓	✓	

IV. SYSTEM MODEL AND PROBLEM STATEMENT

We first present a mathematical representation of the inputs, *i.e.*, the substrate network and the SFC request, then we formally define the problem of orchestrating SFCs with flexible VNF ordering with lowest resource provisioning cost (Flexible SFC orchestration for short).

A. Substrate Network

We represent the substrate network (SN) as an undirected graph $G^s = (N, L)$; where every node $n \in N$ is associated

[2]In the rest of the paper by packet we refer to a Layer-2 frame.

with residual compute resource capacity c_n and internal-switching capacity b_n. Further, every link $l:(i,j) \in L$ is associated with residual bandwidth capacity $b_{i,j}$.

B. Virtual Network Functions

The set of available VNFs is represented by $\mathcal{F}$. Each VNF $f \in \mathcal{F}$ has compute resource requirement (*e.g.*, number of CPU cores) d_f. Additionally, we have a $|\mathcal{F} \times \mathcal{F}|$ matrix $\mathcal{R}$ that represents reorder compatibility between the VNFs. $R_{f,f'} = 1$ if VNF $f \in \mathcal{F}$ and $f' \in \mathcal{F}$ are reorder compatible according to the definition in Section III, 0 otherwise.

C. SFC Request

We represent an SFC request as an directed graph $G^v = (F, E, n_o, n_t)$, where F is the set of VNFs in the SFC. Note that we focus on SFCs that are structured as chains. Considerations for branching in SFCs is left as a future work. The requested chain E is represented by a set of directed virtual links, where each virtual link $e \in E$ consists of a pair of consecutive VNFs, and is associated with bandwidth demand d_e. Further, each SFC is associated with an ingress node $n_o \in N$ and an egress node $n_t \in N$.

D. Problem Statement

Given an SN G^s, set of VNFs $\mathcal{F}$, a reorder compatibility matrix $\mathcal{R}$, and a SFC request G^v:

- Place VNF $f \in F$ on a server $n \in N$.
- Route exactly $|F|-1$ virtual links between the VNFs to form a chain structure.
- Map each virtual link either to a single substrate path or to a server.
- The cost of allocating bandwidth from the network to route the virtual links are minimized subject to the following constraints:
  - Servers and network links cannot be over-committed to accommodate VNFs and the virtual links.
  - A virtual link should be routed on a single path in the network or placed inside a single server in the network.

Flexible SFC orchestration bares some similarity with the well studied Virtual Network Embedding (VNE) problem [26]. However, a fundamental difference between flexible SFC orchestration and VNE is that the set of virtual links to be embedded is given as an input to VNE. Whereas, in case of flexible SFC orchestration this set is not known and is part of the solution instead.

V. *OPT-Khaleesi*: ILP FORMULATION

In this section, we first showcase our solution approach that transforms an SFC to exploit flexibility, followed by OPT-*Khaleesi*, our ILP model that optimally solves the flexible SFC orchestration problem.

A. SFC Transformation

In order to fully exploit the flexibility in a given chain, the model must become aware of every re-order compatible pair of VNFs in this chain. To give the model a complete knowledge of this flexibility, we augment the chain with directed virtual links. The ensemble of the original chain and the augmented directed virtual links represent all possible chains that can be traced. The model then selects the optimal subset of $|F|$-1 virtual links that routes through every VNF in the chain with minimum cost. Given an SFC and a re-order compatibility matrix $\mathcal{R}$, we augment the set E into E' as follows:

- if $\mathcal{R}_{u,v}$ = 1 and $(u \rightarrow v) \in E$, links $(u \rightarrow v$.next) and $(v \rightarrow u)$ are created and added to E'.
- if $\mathcal{R}_{u,v}$ = 1 and $(u \rightarrow v) \notin E$, if $\mathcal{R}_{u,j}$ = 1 $\forall$ $u.next \leq j \leq v.prev$, links $(u \rightarrow v$.next) and $(v \rightarrow u)$ are created and added to E'.
- if $\mathcal{R}_{u,v}$ = 1 and $(u \rightarrow v) \notin E$, if $\mathcal{R}_{j,v}$ = 1 $\forall$ $u.next \leq j \leq v.prev$, links $(u \rightarrow v)$ and $(v \rightarrow u)$ are created and added to E'.

Fig. 2(c) shows the set E' generated for the SFC presented in Fig. 2(a) given the re-order compatibility matrix in Fig. 2(b). Generating the set E' alone is not sufficient; this is because not every subset $\bar{E} \subset E'$ renders a valid chain. Concretely, consider again the chain presented in Fig. 2(c), and recall that VNFs 0 and 1 are not re-order compatible. Here, while chain $\{3 \rightarrow 0 \rightarrow 1 \rightarrow 2\}$ is valid, chain $\{1 \rightarrow 2 \rightarrow 3 \rightarrow 0\}$ is not. Note that both chains are made of a combination of virtual links in E'. Subsequently, we need to ensure that any selected chain structure $\bar{E} \subset E'$ does not violate any semantics. To do so, we introduce Ω, a binary matrix, that indicates whether VNF f can precede function f'.

$$\Omega_{f,f'} = \begin{cases} 1, \text{if } (f,f') \in E \vee (R_{f,f'} = 1 \wedge \text{if } (f,f') \in E'), \\ 0, \text{otherwise} \end{cases}$$

Theorem 1. *Any chain that adheres to Ω is semantically valid.*

Proof. Let $\bar{E} \subset E'$ be semantically invalid. This means that $\bar{E}$ contains a pair of VNFs (f,f') where f cannot precede f'. If f cannot precede f', it means that f and f' are not re-order compatible; thus $R_{f,f'}$ = 0 and by definition $\Omega_{f,f'}$ = $(R_{f,f'} \wedge$ if $(f,f') \in E')$ = 0. This follows that any chain that adheres to Ω is semantically valid. □

B. Decision Variables

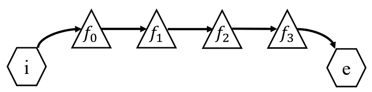
A VNF is placed on a node in the SN, which is represented by the following decision variable:

$$\theta_n^f = \begin{cases} 1 & \text{if VNF } f \in F \text{ is placed on node } n \in N, \\ 0 & \text{otherwise.} \end{cases}$$

The following determines the selection of a virtual link $e \in E'$ for inclusion in the final SFC:

$$z_e = \begin{cases} 1 & \text{if virtual link } e \in E' \text{ is selected,} \\ 0 & \text{otherwise.} \end{cases}$$

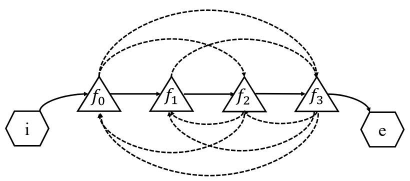
We use the binary variables x_n^e and y_n^e to indicate placement of the origin and destination of a virtual link $e \in E'$ on a



(a) Service Function Chain

	f_0	f_1	f_2	f_3
f_0	0	0	1	1
f_1	0	0	1	1
f_2	1	1	0	1
f_3	1	1	1	0

(b) Re-order Compatibility Matrix



(c) Augmented Set E'

Fig. 2. Semantically Correct Chains

substrate node $n \in N$. Routing of a virtual link $e \in E'$ is determined by the following variable:

$$w^e_{i,j} = \begin{cases} 1 & \text{if } e \in E' \text{ routed on substrate link } (i,j) \in L, \\ 0 & \text{otherwise.} \end{cases}$$

Finally, we use the following variable to derive the ordering between VNFs in the resultant SFC:

$$\delta_{f,f'} = \begin{cases} 1 & \text{if VNF } f \text{ precedes VNF } f' \text{ in the resultant SFC,} \\ 0 & \text{otherwise.} \end{cases}$$

C. Constraints

1) VNF Placement Constraints: Constraint (2) ensures that each VNF is placed on at most a single substrate node. Placement of the origin and destination of each virtual link is ensured by constraints (3) and (4), respectively. Finally, we ensure by (5) that either both the source and destination of a virtual link are placed, or neither.

$$\forall f \in F : \sum_{n \in N} \theta^f_n = 1 \tag{2}$$

$$\forall e \in E', n \in N : x^e_n \leq \sum_{f \in F : o(e) = f} \theta^f_n \tag{3}$$

$$\forall e \in E', n \in N : y^e_n \leq \sum_{f \in F : t(e) = f} \theta^f_n \tag{4}$$

$$\forall e \in E' : \sum_{n \in N} x^e_n - \sum_{n \in N} y^e_n = 0 \tag{5}$$

2) SFC Selection Constraints: The first and last VNF in the resultant SFC is determined by (6) and (7), respectively. Then, (8) ensures that a virtual link is routed *iff* both of its endpoints are placed. Constraints (9) and (10) ensure that every VNF in the resultant SFC is traversed exactly once. We ensure that exactly $|F| - 1$ virtual links are placed by (11). Finally, (12) is used to break loops between virtual links.

$$\forall \bar{e} \in E' : \{o(\bar{e}) = n_o\}, \forall f \in F : \sum_{n \in N} y^{\bar{e}}_n \leq 1 - \sum_{\substack{e \in E' : \{e \neq \bar{e}\}, \\ \{t(e) = f\}}} z_e \tag{6}$$

$$\forall \bar{e} \in E' : \{t(\bar{e}) = n_t\}, \forall f \in F : \sum_{n \in N} x^{\bar{e}}_n \leq 1 - \sum_{\substack{e \in E' : \{e \neq \bar{e}\}, \\ \{o(e) = f\}}} z_e \tag{7}$$

$$\forall e \in E' : z_e \leq \frac{1}{2}\left(\sum_{n \in N} x^e_n + \sum_{n \in N} y^e_n\right) \tag{8}$$

$$\forall f \in F : \sum_{e \in E' : (o(e) = f)} \sum_{n \in N} x^e_n \leq 1 \tag{9}$$

$$\forall f \in F : \sum_{e \in E' : (t(e) = f)} \sum_{n \in N} y^e_n \leq 1 \tag{10}$$

$$\sum_{e \in E'} z_e = |F| - 1 \tag{11}$$

$$\forall e \in E', e' \in E' : \{o(e) = t(e') \wedge t(e) = o(e')\} : z_e + z_{e'} \leq 1 \tag{12}$$

3) Substrate Capacity Constraints: (13), (14), and (15) represent the server capacity, internal switching capacity, and substrate link capacity constraints, respectively.

$$\forall n \in N : \sum_{f \in F} \theta^f_n \cdot d_f \leq c_n \tag{13}$$

$$\forall f \in F : \sum_{e \in E'} (x^e_n \cdot y^e_n) \cdot d_e \leq b_n \tag{14}$$

$$\forall (i,j) \in L : \sum_{e \in E'} \sum_{(i,j) \in L} w^e_{i,j} \cdot d_e \leq b_{i,j} \tag{15}$$

4) SFC Routing Constraints: Constraint (16) represents the flow conservation for mapping the virtual links. We use (17) to determine the ordering of VNFs in the resultant SFC. Finally, (18) ensures that the order of VNFs preserves the SFC's semantics.

$$\forall i \in N, e \in E' : \sum_{j : (i,j \in L)} w^e_{i,j} - \sum_{j : (j,i) \in L} w^e_{j,i} = x^e_i - y^e_i \tag{16}$$

$$\forall f \in F, \forall f' \in F, \forall f'' \in F : \delta_{f,f"} \geq (\delta_{f,f'} \cdot \delta_{f',f"}) + z_{\substack{e : \{o(e) = f, \\ t(e) = f"\}}} \tag{17}$$

$$\forall f \in F, \forall f' \in F : \delta_{f,f'} \leq \Omega_{f,f'} \tag{18}$$

Note that (14) and (17) contain product of two integer variables, which renders the model non-linear. However, the product of two integer variables can be linearized as follows: For Constraint (14), we introduce a new variable $g^e_n \in \{0, 1\}$

such that:

$$g_n^e \leq x_n^e \quad (19)$$

$$g_n^e \leq y_n^e \quad (20)$$

$$g_n^e \geq x_n^e + y_n^e - 1 \quad (21)$$

Similarly, we linearize Constraint (17) by introducing a new variable $q_{f,f',f''} \in \{0,1\}$ such that

$$q_{f,f',f''} \leq \delta_{f,f'} \quad (22)$$

$$q_{f,f',f''} \leq \delta_{f',f''} \quad (23)$$

$$q_{f,f',f''} \geq \delta_{f,f'} + \delta_{f',f''} - 1 \quad (24)$$

D. Objective Function

Our objective is to embed the SFC while minimizing the incurred cost in terms of bandwidth consumption.

$$\text{minimize} \sum_{e \in E'} \sum_{(i,j) \in L} w_{i,j}^e$$

E. Hardness of the Problem

Theorem 2. *OPT-Khaleesi is NP-Complete.*

Proof. Given a graph G^s=(N,L), where c_n = 1 $\forall$ $n \in N$ and b_l = 1 $\forall$ $l \in L$. We transform G^s into G'^s=(N',L') by adding an auxiliary node of capacity 0 between every pair (i,j) $\in$ L. G'^s thus represents an SN where some nodes are servers and the rest are network nodes. Now assume that we have a *chaotic* SFC request of size N with 1 unit demand for each VNF in the SFC and d_f = 1. A chaotic SFC refers to an SFC where all pair of VNFs are re-order compatible. Solving the Hamiltonian path problem in G^s corresponds to finding a path that spans every node in N. This is exactly solving the flexible service chaining problem in G'^s since the Hamiltonian path will span N compute nodes in G^s (the size of the chain). Conversely, solving the flexible service chaining in G'^s would mean that we have found a chain that spans N compute nodes. This chain in G'^s corresponds to a Hamiltonian path in G^s. Since computing Hamiltonian path is NP-Complete, therefore a special case of our problem (*i.e.*, placement of a chaotic chain of size N) is also NP-Complete. Therefore, by restriction, *OPT-Khaleesi* is NP-complete. ☐

VI. *FAST-Khaleesi*: Heuristic Solution

To overcome the computational complexity of *OPT-Khaleesi*, we propose *FAST-Khaleesi*, a heuristic that performs flexible chaining, VNF placement, and routing with the objective to minimize bandwidth footprint. *FAST-Khaleesi* consists of 4 Steps, and is designed to select the chain that encourages more virtual links to be routed internally, which in turn reduces the number of inter-server switching.

- **Step 1: Generate all SFCs** First, given the set E', we trace all possible chains that do not violate any semantics (*i.e.*, respect Ω) by using backtracking. We denote this set as $\mathcal{S}$. Generating all possible chains for a SFC where all VNFs are re-order compatible may yield an exponential number of combinations $O(|F|!)$ in the worst case. However, in practice, the size of SFC does not exceed 6 in a DC [27]. Moreover, not all pairs of VNFs are typically reorder compatible with each other. Therefore, in practice the actual number of valid chains is far less than the worst case.
- **Step 2: Find Candidate Servers** Step 2 consists of finding a list of candidate servers. A candidate server is a server with sufficient CPU resources to accommodate any VNF. Once the list of candidate servers is obtained, we compute the shortest path between each candidate server $n \in \bar{N}$ and the ingress n_o and egress n_t nodes of the chain, respectively. We denote these distances as x_o and x_t, respectively. Subsequently, we compute the ratio r_n = $\frac{c_n}{x_o+x_e}$ for every node $n \in \bar{N}$. By sorting the servers in decreasing order of ratio r, we prioritize the servers with highest capacity and proximity to the chain's ingress and egress nodes. This will also potentially reduce the bandwidth footprint, as the VNFs will be placed close to the origin and sink of the chain at hand. The time complexity of Step 2 is O (2 $\cdot$ $\bar{N}$ $\cdot$ (L + $N$log$N$) + $\bar{N}$log$\bar{N}$) $\leq$ O(N^2log$N$) by dropping lower-order terms.
- **Step 3: VNF Placement** The VNF placement is performed for every chain as follows: First, for every candidate server $n \in \bar{N}$ and for every VNF $f \in F$, we

Algorithm 1: FAST-*Khaleesi* Algorithm

```
Input: $G^s$ = ($N$,$L$), $G^v$ = ($F$, $E$, $n_o$, $n_t$), $E'$
Output: NF Placement and Chain Routing Solution $m$
1  function FAST-Khaleesi
2      Step 1: Generate all Valid Chains $S$
3      Step 2: Find all candidate servers in $N$
4      $\bar{N}$ = {}
5      forall $n \in N$ do
6          if ($c_n < \min_{\forall f \in F} d_f$) then
7              continue
8          $\bar{N}$ = $\bar{N} \cup \{n\}$
9          $r_n$ = |shortestPath($n, n_0$)|+|shortestPath($n, n_t$)|
10     SortDescendingOrder($\bar{N}$,$\frac{c}{r}$)
11     $\mathcal{M}$ = {}; /*Initialize an empty solution set*/
12     forall $s = (F, \bar{E}) \in S$ do
13         Step 3: VNF Placement
14         $\bar{m}$ = {} /*Initialize an empty solution*/
15         $V$ = $F$
16         while ($|V| > 0$) do
17             $\bar{F}$ = FindMinOpCost$_{\forall v \in V}$($s$,$\bar{N}$)
18             $\bar{m}$.nmap = $\bar{m}$.nmap $\cup$ ($\bar{F}$,$n$)
19             $V$ = $V$ - $\bar{F}$
20             $\bar{E}$ = $\bar{E}$ - GetInternallySwitchedLinks($\bar{F}$)
21         Step 4: Virtual Link Routing
22         forall ($e \in \bar{E}$) do
23             $\bar{m}$.emap = $\bar{m}$.emap $\cup$ Dijkstra($e$)
24         $\mathcal{M}$ = $\mathcal{M}$ $\cup$ {$\bar{m}$}
25     $m$ = FindLowestCostSol($\mathcal{M}$)
26     return $m$
```

compute an opportunity cost of placing f along with a subset of VNFs $\bar{F}$ on n. The subset $\bar{F} \cup f$ with the lowest opportunity cost is chosen and mapped on node n. Here, opportunity cost refers to the number of virtual links that require inter-server switching as result of placing $\bar{F} \cup f$ on n. This cost is reduced by maximizing the number of virtual links with both end points placed on the same server. Hence, to minimize the opportunity cost for every VNF f, we traverse the chain s starting at f, and every time we find a pair of adjacent VNFs that can be packed in n without violating its capacity we add them to $\bar{F}$. This iteration is repeated until the placement of all VNFs is settled. At every iteration, the set of virtual links that require inter-server routing is updated. The time complexity of Step 3 is $O(|S| \cdot |\bar{N}| \cdot |F|^2)$.
- **Step 4: Inter-Server Routing** Finally, for every placement solution generated in Step 3, Dijkstra's shortest path algorithm is performed to route the set of virtual links whose end points are distributed in different servers. The time complexity of Step 4 is $O(|S| \cdot (|L| + |N|\log|N|))$.

Once Steps 3 and 4 are performed for every SFC, the algorithm terminates, and the mapping solution with the lowest total bandwidth consumption is returned. If no feasible mapping solution can be found for any chain, the SFC is rejected.

VII. Performance Evaluation

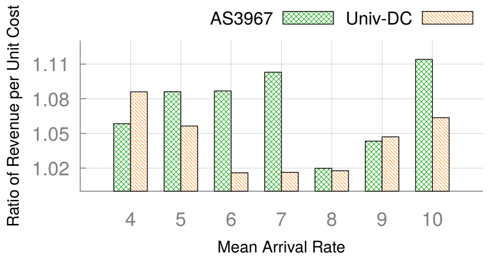
We evaluate our proposed solutions in the following scenarios: (i) evaluating the benefits of flexible VNF ordering in SFCs (Section VII-C), and (ii) performance comparison between *FAST-Khaleesi* and *OPT-Khaleesi* (Section VII-D). Before presenting the results, we describe our simulation setup in Section VII-A and the evaluation metrics in Section VII-B.

A. Simulation Setup

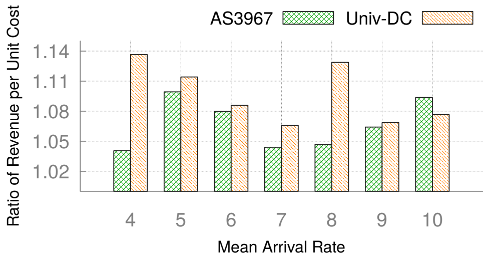
1) Testbed: We implemented *OPT-Khaleesi* and *FAST-Khaleesi* using IBM ILOG CPLEX 12.5 Java libraries and Java, respectively. Our testbed consists of machines with hyper-threaded Intel 8×10-core Xeon E7-8870 CPU and 1TB of memory. We developed an in-house discrete event simulator to simulate the arrival and departure of SFCs on an SN.

2) Topology: We used the following two real topologies, representing two different scenarios and scales for our evaluation: (i) Univ-DC: a university data center topology with 23 nodes and 42 links from [28]; and (ii) AS3967: a moderate size ISP topology with 79 nodes and 147 links from Rocketfuel dataset [29]. (i) represents a data center network, which typically has high server density compared to an ISP's backbone network such as (ii). To represent this diversity in server density we augmented (i) and (ii) with 144 and 64 servers, respectively, each with 8 CPU cores.

3) Traffic Data: For Univ-DC topology, we used real traffic traces from the same data center [28] to generate SFC request between pairs of edge switches. For AS3967, no real traffic traces are available, hence, we resorted to generating synthetic traffic. We used FNSS tool [30] and generated time varying traffic by following the distribution presented in [31].



(a) Orchestration of Flexible vs Rigid SFCs



(b) Impact of considering flexible VNF ordering in existing SFC orchestration algorithm [2]

Fig. 3. Benefits of Flexible VNF ordering in SFC

4) Middlebox Data: We selected a set of six VNFs from the ones listed in Table II and computed the reorder compatibility matrix accordingly. We randomly chained subsets of the selected VNFs to generate SFCs with lengths between 3 and 6. Middlebox CPU requirements were obtained from the research literature and available vendor data sheets [2], [32]. SFC arrival and departure was generated following a Poisson distribution with mean arrival rate varied between 4 - 10 SFCs per 100 time unit. Mean life time of these SFCs were set to 1000 time units. The simulation was run for a total of 10000 time units and included 400 - 960 SFC requests. Note that the dataset and parameters chosen for evaluation are in accordance with relevant research literature [3], [26].

B. Evaluation Metrics

1) Acceptance Ratio: The ratio of number of admitted SFC requests to the total number of SFC requests.

2) Embedding Path Length: Embedding path length is the sum of lengths of all the paths used for routing all inter-NF links in an SFC.

3) Revenue per unit cost: Revenue is computed as a function that is proportional to an SFCs total resource requirement. Revenue earned per unit cost is calculated by dividing revenue from an SFC by the SFC's embedding cost.

C. Benefits of Flexibility in SFCs

We demonstrate the benefits of flexible VNF ordering in SFCs by evaluating the following two scenarios. First, we compare results of optimally orchestrating flexible SFCs with that of orchestrating non-flexible or rigid SFCs. In the second scenario, we empirically evaluate how much benefit we can get by feeding all possible SFCs stemming from a flexible SFC to an existing orchestration algorithm from the literature.

The goal is to evaluate how much benefit we can get even without explicitly considering flexible VNF ordering in an existing SFC orchestration algorithm. For that purpose, we use the dynamic programming algorithm from [2].

For the first scenario, we implemented an ILP similar to [2] for comparison. From the results, we did not observe much of a difference between the two approaches in terms of acceptance ratio for different arrival rates. However, the number of accepted SFCs does not say much about the types of SFCs that were accepted. Therefore, we further analyzed the solutions and computed the revenue earned per unit cost for different arrival rates and present the result in Fig. 3(a). More specifically, we present the ratio of revenue earned per unit cost for flexible and rigid cases. The flexible case always yielded more revenue per unit embedding cost to an extent of 11% compared to the rigid cases.

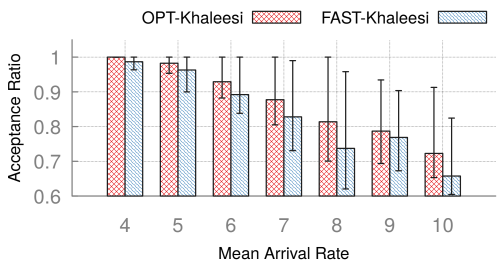
For the second scenario, we take a black box approach. Instead of modifying the dynamic programming algorithm from [2], we executed the algorithm for all valid SFCs that can be traced from an SFC request. We keep the result with the lowest cost for each SFC request. We compare the results from this setting with that from executing [2] on non-flexible SFCs. The results are presented in Fig. 3(b). We observe that even without explicitly considering and exploiting flexibility in SFCs, there is about 10% improvement in revenue earned per unit cost on average for [2].

The takeaway from this study is that flexibility in SFCs can yield more revenue per unit cost even for an SFC orchestration algorithm not designed to handle flexible SFCs. An intuition behind such result is that the flexibility in an SFC leads to reordering of some of the VNFs to accept some more resource demanding ones compared to the non-flexible case.

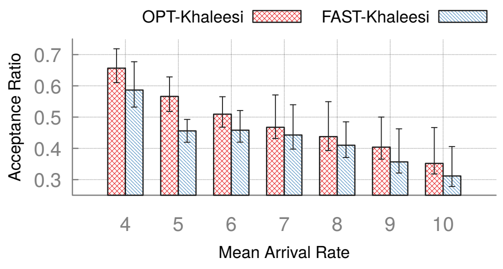
D. *FAST-Khaleesi vs. OPT-Khaleesi*

1) Acceptance Ratio: We present the cumulative acceptance ratio of *OPT-Khaleesi* and *FAST-Khaleesi* for both SN topologies in Fig. 4 with 5th and 95th percentile error bars. We consider the first 1000 time units of simulation as the warm up period and discard results from that period. For a compute resource constraint environment as in AS3967, the performance gap between the heuristic and the optimal is larger than Univ-DC topology. Nonetheless, we found the heuristic to accept no greater than 20% less SFC requests on average compared to the optimal solution.

2) Mean Embedding Path Length: Our cost function is proportional to the embedding path length for an SFC. Therefore, we compare the mean embedding path length across all SFC requests to gain an estimate as to how much far off is the heuristic from the optimal solution. The results for both of the topologies are presented in Fig. 5. For the data center topology where network diameter is relatively smaller compared to the ISP topology, the heuristic's mean embedding path was within 20% of that of the optimal solution. However, on a network with larger diameter such as the ISP topology we used, this stretch increased up to $\approx 2\times$ the optimal solution.

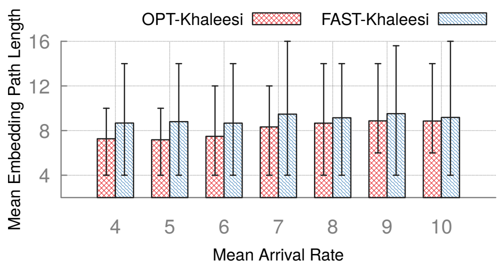


(a) Univ-DC Topology

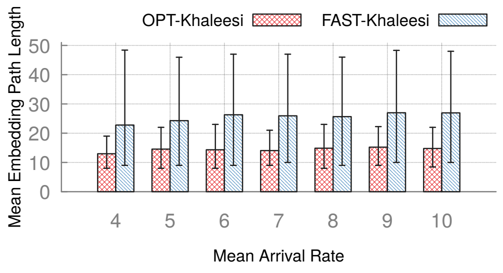


(b) AS3967 Topology

Fig. 4. Acceptance Ratio vs. Load



(a) Univ-DC Topology

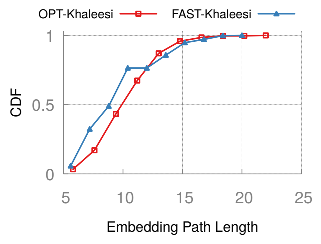


(b) AS3967 Topology

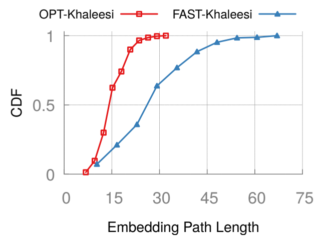
Fig. 5. Comparison of Mean Embedding Path Length

We also show the Cumulative Distribution Function (CDF) of embedding path length of one fixed arrival rate in Fig. 6. The purpose is to show the breakdown of the length distribution and try to understand the long errorbars in Fig. 5. As we can see, the heuristic demonstrates a long tailed CDF. This long tail is due to the heuristic's shortcoming of finding a good solution when the network is very close to saturation.

3) Execution Time: Execution time for *OPT-Khaleesi* and *FAST-Khaleesi* for embedding a single SFC request is presented in Table III. Note that execution time for Univ-DC is higher than that of AS3967. This is because, majority of the execution time in both of the solutions is spent to find



(a) Univ-DC Topology



(b) AS3967 Topology

Fig. 6. Mean Embedding Path Length vs. Load

TABLE III
AVERAGE EXECUTION TIME PER SFC REQUEST

Topology	*OPT-Khaleesi*	*FAST-Khaleesi*
Univ-DC	4538ms	61ms
AS3967	1920ms	43ms

suitable placement of VNFs on the servers. Recall that the data center topology has higher number of servers, hence, higher execution time despite being smaller than AS3967.

VIII. DISCUSSION

In light of the above, we clearly can see potential advantages of flexible structures over the rigid ones. However, the degree of flexibility that might be available in a network not only depends on the types of VNFs deployed, but also on the operator's policies and customer requirements. There are certain NFs, which by policy might not be flexible at all. For instance, a network's policy or a customer's requirement may govern that all incoming and outgoing traffic must traverse through a firewall. In such cases, even if the VNF is reorder compatible with others, it cannot be considered for such flexibility. However, there are indeed some NFs that are deployed for performance enhancement purposes such as a WAN Optimizer, Video Transcoder, Traffic Shaper *etc*., that may have less strict requirement on their order. Our solution can also work with such operator policies. In such cases, where we have additional constraints imposed by policies, setting appropriate entries in $\mathcal{R}$ before running the optimizations should be sufficient.

IX. CONCLUSION

In this paper, we have taken a first step towards studying the impact of flexibility in SFCs and also how such flexibility can be leveraged for resource allocation. We present the first quantifiable results showing the potential benefits of flexible SFCs over rigid ones. Our results show that indeed there is improvement in revenue per unit cost, however, the significance of this improvement can be better explained when it is translated into actual monetary values. To the best of our knowledge, we are the first to propose an ILP-based optimal solution to the problem. We also propose a heuristic that performs within $2\times$ of the optimal solution on average while executing orders of magnitude faster.

Flexibility in SFC can be leveraged in other scenarios as well. For instance, in the event of substrate node or link failure, flexibility can be leveraged for restoring failed SFCs while minimizing backup footprint. Another interesting direction is to investigate how flexibility in SFC can be applied for re-optimizing resource allocation to alleviate bottlenecks.

REFERENCES

[1] "Network Functions Virtualisation - Introductory White Paper," Oct 2012. [Online]. Available: https://portal.etsi.org/nfv/nfv_white_paper.pdf
[2] F. Bari *et al.*, "Orchestrating virtualized network functions," *IEEE Trans. on Net. and Service Management*, vol. 13, no. 4, pp. 725–739, 2016.
[3] J. G. Herrera *et al.*, "Resource allocation in nfv: A comprehensive survey," *IEEE Trans. on Net. and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.
[4] S. Mehraghdam *et al.*, "Specifying and placing chains of virtual network functions," in *Proc. of IEEE CloudNet*, 2014, pp. 7–13.
[5] ——, "Placement of services with flexible structures specified by a yang data model," in *Proc. of IEEE NetSoft*, 2016.
[6] S. Dräxler *et al.*, "Specification, composition, and placement of network services with flexible structures," *International Journal of Network Management*, vol. 27, no. 2, 2017.
[7] Y. Zhang *et al.*, "Parabox: Exploiting parallelism for virtual network functions in service chaining," in *Proc. of ACM SOSR*, 2017, pp. 143–149.
[8] C. Sun *et al.*, "Nfp: Enabling network function parallelism in nfv," in *Proc. of ACM SIGCOMM*, 2017, pp. 43–56.
[9] S. Ayoubi *et al.*, "A cut-and-solve based approach for the vnf assignment problem," *IEEE Trans. on Cloud Computing*, 2017.
[10] H. Moens *et al.*, "Vnf-p: A model for efficient placement of virtualized network functions," in *Proc. of CNSM*, 2014, pp. 418–423.
[11] B. Addis *et al.*, "Virtual network functions placement and routing optimization," in *IEEE CloudNet*, 2015, pp. 171–177.
[12] M. C. Luizelli *et al.*, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *Proc. of IFIP/IEEE IM*, 2015, pp. 98–106.
[13] R. Cohen *et al.*, "Near optimal placement of virtual network functions," in *Proc. of IEEE INFOCOM*, 2015, pp. 1346–1354.
[14] W. Ma *et al.*, "Traffic aware placement of interdependent nfv middleboxes," in *Proc. of IEEE INFOCOM*, 2017, pp. 1–9.
[15] H. Song, "Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane," in *Proc. of ACM HotSDN*, 2013, pp. 127–132.
[16] D. Joseph *et al.*, "Modeling middleboxes," *IEEE network*, vol. 22, no. 5, 2008.
[17] A. Panda *et al.*, "Verifying reachability in networks with mutable datapaths." in *Proc. of USENIX NSDI*, 2017, pp. 699–718.
[18] S. K. Fayaz *et al.*, "Buzz: Testing context-dependent policies in stateful networks." in *Proc. of USENIX NSDI*, 2016, pp. 275–289.
[19] P. Kazemian *et al.*, "Header space analysis: Static checking for networks." in *Proc. of USENIX NSDI*, 2012, pp. 113–126.
[20] W. Wu *et al.*, "Automatic synthesis of nf models by program analysis," in *Proc. of ACM HotSDN*, 2016, pp. 29–35.
[21] ——, "Network function modeling and its applications," *IEEE Internet Computing*, vol. 21, no. 4, pp. 82–86, 2017.
[22] J. Sherry *et al.*, "Making middleboxes someone else's problem: network processing as a cloud service," *ACM SIGCOMM Computer Comm. Rev.*, vol. 42, no. 4, pp. 13–24, 2012.
[23] S. W. Brim *et al.*, "Middleboxes: Taxonomy and issues," 2002.
[24] N. Freed, "Behavior of and requirements for internet firewalls," 2000.
[25] J. Martins *et al.*, "Clickos and the art of network function virtualization," in *Proc. USENIX NSDI*, 2014, pp. 459–473.
[26] A. Fischer *et al.*, "Virtual network embedding: A survey," *IEEE Comm. Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.
[27] S. Kumar *et al.*, "Service function chaining use cases in data centers," *IETF SFC WG*, 2015.
[28] T. Benson *et al.*, "Network traffic characteristics of data centers in the wild," in *Proc. of ACM IMC*, 2010, pp. 267–280.
[29] N. Spring *et al.*, "Measuring isp topologies with rocketfuel," *ACM SIGCOMM Computer Comm. Rev.*, vol. 32, no. 4, pp. 133–145, 2002.
[30] L. Saino *et al.*, "A toolchain for simplifying network simulation setup," in *Proc. of SIMUTools*, 2013, pp. 82–91.
[31] A. Nucci *et al.*, "The problem of synthetically generating ip traffic matrices: initial recommendations," *ACM SIGCOMM Computer Comm. Rev.*, vol. 35, no. 3, pp. 19–32, 2005.
[32] "Wanos wan optimizer admin guide - hardware requirements." [Online]. Available: http://wanos.co/docs/docs/wanos-admin-guide/getting-started/hardware-requirements/