Traffic Aware Placement of Interdependent NFV Middleboxes

Wenrui Ma, Oscar Sandoval*, Jonathan Beltran, Deng Pan, and Niki Pissinou
School of Computing and Information Sciences, Florida International University, Miami, FL
*Department of Computer Science, Univeristy of Virginia, Charlottesville, VA
{wma006, jbelt021, pand, pissinou}@fiu.edu, *oks2vd@virginia.edu

***Abstract*—Network function virtualization enables flexible implementation of network functions, or middleboxes, as virtual machines running on standard servers. However, the flexibility also creates a challenge for efficiently placing such middleboxes, due to the availability of multiple hosting servers, capability of middleboxes to change traffic volumes, and dependency between middleboxes. In this paper, we address the optimal placement challenge of NFV middleboxes, and propose solutions for middleboxes of different traffic changing effects and with different dependency relations. First, we formulate the Traffic Aware Placement of Interdependent Middleboxes problem as a graph optimization problem. When the flow path is predetermined, we design optimal algorithms to place a non-ordered or totally-ordered middlebox set, and propose an efficient heuristic for the general scenario of a partially-ordered middlebox set after proving its NP-hardness. When the flow path is not predetermined, we show that the problem is NP-hard even for a non-ordered middlebox set, and propose a traffic and space aware routing heuristic. We have evaluated the proposed algorithms using large scale simulations and prototype experiments, and present extensive evaluation results to demonstrate the effectiveness of our design.**

***Index Terms*—NFV, SDN, middlebox**

I. Introduction

Network function virtualization (NFV) transforms the implementation of network functions, also called middleboxes, from proprietary hardware appliances to virtual machines (VMs) running on industry standard servers [27]. Leveraging the underlying virtualization technology, VM-based software middleboxes bring many benefits that were not previously available, such as accelerated time-to-market, reduced hardware and operation cost, improved security, and elastic scalability [6].

The flexibility of VMs also poses challenges for efficient NFV implementation. In particular, traditional hardware middleboxes are deployed at fixed locations in the network, and leave no choice of service locations. In an NFV network, each switch may have one or more attached NFV servers with standard hardware that can host VMs of arbitrary network functions [27]. It is thus possible to optimize the network performance by carefully selecting the location to place a software middlebox among multiple candidate servers. Improper placement decisions may cause inefficient flow paths and traffic jam.

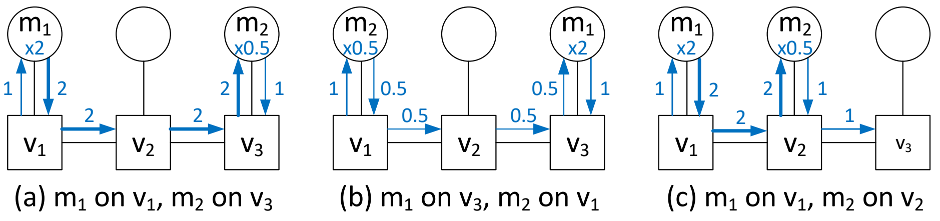
Furthermore, the NFV service location challenge is complicated by the traffic changing effects of middleboxes. Unlike switches and routers that forward traffic without changing its volume, middleboxes may change the traffic volumes of processed flows, and may do it in different ways. For example, the Citrix CloudBridge WAN optimizer compresses traffic before sending it to the next hop, and may reduce the traffic volume by up to 80% [4]. On the other hand, the BCH(63,48) encoder, used for satellite communications signaling messages, increases the traffic volume by 31% due to the checksum overhead [21].

We use the following example to illustrate the traffic changing effects of middleboxes. Consider a network of three nodes v_1, v_2 and v_3, and two links (v_1, v_2) and (v_2, v_3). Each node has an attached NFV server (denoted as a circle in Fig. 1), and each server can host a single middlebox. A flow f starts at v_1 and ends at v_3, whose initial traffic rate is 1. Two middleboxes m_1 and m_2 need to be applied to f. m_1 will double the traffic rate, while m_2 will decrease it by 50%. By placing m_1 on v_1 and m_2 on v_3, the loads of links (v_1, v_2) and (v_2, v_3) will be $1 \times 2 = 2$, as shown in Fig. 1(a). However, by placing m_1 on v_3 and m_2 on v_1, the loads of both links are reduced to $1 \times 0.5 = 0.5$, as shown in Fig. 1(b).

The placement of middleboxes is also constrained by the dependency relation that may or may not exist between middleboxes [26]. For instance, an IPSec decryptor is usually placed before a NAT gateway [3], while a VPN proxy can be placed either before or after a firewall [5]. In the above example, if there is a constraint for m_1 to be applied before m_2, then the placement scheme in Fig. 1(b) would violate the constraint. However, by placing m_1 on v_1 and m_2 on v_2, we can still reduce the load of link (v_2, v_3) from 2 to 1 as in Fig. 1(c), in contrast to the case in Fig. 1(a).

In this paper, we study the optimal placement of NFV middleboxes. Compared with the existing works in Section II, this paper proposes comprehensive solutions that consider different traffic changing effects of middleboxes as well as different types of middlebox dependency relations. Our design utilizes the emerging Software-Defined Networking (SDN) technology as the implementation platform, because it enables efficient optimization by decoupling the network control plane and data plane. An SDN based prototype is implemented to validate our design.

The proposed algorithms achieve load balancing for large-size and long-duration elephant flows [9]. Our design philosophy is to prevent different elephant flows from sharing the



m_1 x2
m_2 x0.5
1
2
2
2
2
1
v_1
v_2
v_3
(a) m_1 on v_1, m_2 on v_3
m_2 x0.5
m_1 x2
1
0.5
0.5
0.5
0.5
1
(b) m_1 on v_3, m_2 on v_1
m_1 x2
m_2 x0.5
1
2
2
2
1
1
(c) m_1 on v_1, m_2 on v_2

Fig. 1: Traffic changing effects of middleboxes.

same middlebox to avoid resource contention and congestion, but instead let mice flows utilize the spare processing capacity of already deployed middleboxes. Elephant flows can be statically determined based on the application types, such as data backup or VM migration, or dynamically detected using existing techniques in the literature [9], [23]. The solutions for mice flows will not be the focus of this paper.

Our main contributions in this paper are summarized as follows.

1) We formulate the Traffic Aware Placement of Interdependent Middleboxes (TAPIM) problem, considering in particular a generalized partial order for the middlebox dependency relation, as a graph optimization problem with the objective to load-balance the network.
2) For topologies with predetermined paths, such as the tree, we design optimal algorithms for the special case when the middlebox set is a non-ordered or totally-ordered one. For the general case when the dependency relation is a partial order, we show that the TAPIM problem is NP-hard by reduction from the Clique problem, and propose an efficient heuristic to convert a partially-ordered set to a totally-ordered one.
3) For topologies without predetermined paths, we prove that the TAPIM problem is NP-hard even for a non-ordered middlebox set by reduction from the Hamiltonian Cycle problem. Our proposed solution then works in two steps: first finding a path with enough resources to host all the middleboxes, and then placing the middleboxes on the given path.
4) We have implemented the proposed algorithms in the NS-3 simulator and a SDN based prototype, and present extensive simulation and experiment results to demonstrate the effectiveness of our design.

The remaining of this paper is organized as follows. Section II provides a brief overview of related works. Section III formulates the TAPIM problem. Sections IV and V solve the TAPIM problem with and without predetermined flow paths, respectively. Section VI presents experiment and simulation results. Finally, Section VII concludes the paper.

II. Related Works

As the promising platform for network functioning provisioning, NFV has attracted significant research attention. Multiple efforts have been focusing on designing efficient hardware and software architectures. Sekar et al. propose an NFV architecture named CoMb [27] that explores consolidation opportunities to reduce network provisioning cost and load skew. Anderson et al. propose the xOMB [14] architecture for building scalable and extensible middleboxes. Hwang et al. propose the NetVM [15] network virtualization platform, to dynamically scale, deploy, and reprogram network functions. ClickOS [16] is a high-performance and virtualized software middlebox platform, which enables hundreds of concurrent virtual network functions without significant overhead in packet processing. This work can benefit from the above research advances by utilizing those NFV architectures to implement our design.

It has been recognized as a challenge to implement a chain of network functions under certain dependency constraints. Moens and Turck present an Integer Linear Program (ILP) model named VNF-P [22] for resource allocation of NFV service chains, and evaluate it with a case study simulation. Mehraghdam et al. define a model for formalizing the chaining of network functions using a context-free language [26], and allocate network resources by solving a Mixed Integer Quadratically Constrained Program (MIQCP). Rami et al. formulate the NFV location problem [24], and propose near optimal approximation algorithms for NFV resource allocation without considering network function dependency. Li et al. propose the NFV-RT [19] resource provisioning system that uses a linear programming approach to maximize the number of requests for each service chain, and validate the design by simulation and emulation. The formal model defined in this work differs from the above ones in considering different middlebox traffic changing effects and dependency relations. In addition to the formal model, this work also proposes practical algorithms, and conducts evaluations based on prototype implementations as well as large scale simulations.

Due to its capability of global optimization, SDN is commonly adopted as the control protocol to maneuver traffic in an NFV network. For example, SIMPLE [28] is an SDN-based policy enforcement layer for efficient middlebox-specific traffic steering. Multiple architectures [13], [17] are proposed to integrate SDN and NFV for efficient implementation of service chains. To correctly reason and enforce network-wide policies, Fayazbakhsh et al. propose an extended SDN architecture called FlowTags [25], in which middleboxes add packet tags to provide the necessary context for systematic policy enforcement. Our work differs from the above ones by not only implementing the correct policies through SDN, but also considering the middlebox traffic changing effects and different dependency relations.

To the best of our knowledge, no existing research has focused on the traffic changing effects of middleboxes, except our preliminary work [20], which did not consider dependency relations between middleboxes. Compared with the extensively studied problem on general VM placement [18], this work addresses not only the VM placement but also flow routing issue, and targets network load balancing instead of resource utilization.

III. Problem Statement

In this section, we formulate the Traffic Aware Placement of Interdependent Middleboxes (TAPIM) problem.

Consider a network represented by a directed graph $G = (V, E)$. Each node $v \in V$ may have one or more attached NFV servers, and its space capacity is denoted as $sc[u]$[1] ≥ 0, i.e., the maximum number of middleboxes to host. For simplicity, we assume that each middlebox needs one space, and additional processing power can be achieved by multiple load-balanced middlebox instances [10]. A link $(u, v) \in E$ has a bandwidth capacity $bc[u, v] \geq 0$, i.e., the amount of available bandwidth. For easy representation, we define connectivity by $connect[u, v] = 1$ if $bc[u, v] > 0$. The existing load of the link is denoted as $load[u, v]$.

For route calculation, each link $(u, v) \in E$ is assigned a weight, denoted as $weight(u, v, l)$, which is a non-decreasing function of the link load l, i.e., $\forall l \leq l', weight(u, v, l) \leq weight(u, v, l')$. A broad category of weight functions satisfy the non-decreasing requirement, such as the ones used by the popular Cisco EIGRP [2] and OSPF [7] protocols. The non-decreasing link weight function helps load-balance network traffic when the routing protocol aims to minimizes the path cost, which is defined as the weight sum of all the path links.

An elephant flow f is defined as a 4-tuple (src, dst, t, M), in which $src \in V$ is the source node, $dst \in V$ is the destination node, t is the initial traffic rate when f arrives at the ingress switch of the network, and M is the set of required middleboxes. Multiple flows are processed one at a time, because flows tend not to arrive at the exactly same time in reality. Particularly in an SDN network, even if multiple flows arrive simultaneously, the SDN controller will have to process them one by one.

Each middlebox $m \in M$ has an associated traffic changing factor $ratio[m] \geq 0$, which is the ratio of the traffic rate of a flow after and before being processed by m. The *dependency* relation $\leftarrow$ is defined as a strict partial order on M that is

1) Irreflexive: $m \not\leftarrow m$,
2) Transitive: $m \leftarrow m'$ and $m' \leftarrow m''$ then $m \leftarrow m''$, and
3) Asymmetric: $m \leftarrow m'$ then $m' \not\leftarrow m$.

If $m \leftarrow m'$, we say that the middlebox m' depends on m, or intuitively m' should be applied after m. If m depends on no other middlebox, i.e., $\forall m' \in M, m' \not\leftarrow m$, we say that m has no dependence. For easy representation, define $depend[m, m'] = 1$ if $m \leftarrow m'$, and 0 otherwise.

[1]We use square brackets [] to denote properties or known values, and round brackets () denote to functions or variables.

When the flow f enters the network, a multi-hop path, denoted as *route*, will be assigned for the flow, which is a decision variable defined as

$$route(v, i) = \begin{cases} 1, & \text{if } v \in V \text{ is the } i^{th} \text{ hop on the path.} \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

We define i to start from one, and denote the last hop number as n for convenience. Note that repeating nodes are allowed on the path to enable more general solutions, i.e., $\exists v, i \neq i' : route(v, i) = 1$ and $route(v, i') = 1$. To avoid performance degradation for TCP flows, a flow is not allowed to be split among two paths [11].

In addition, a placement scheme, denoted as *place*, will determine the location for each middlebox $m \in M$, which is a decision variable defined as

$$place(m, i) = \begin{cases} 1, & \text{if } m \in M \text{ is placed on the } i^{th} \text{ hop.} \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

Use $t_{in}(i)$ and $t_{out}(i)$ to denote the incoming and outgoing traffic rate of flow f at the i^{th} hop on the path, respectively. If f is processed by a single middlebox m at the i^{th} hop, then $t_{out}(i) = t_{in}(i)ratio[m]$. Note that the incoming traffic rate at the flow source is the initial traffic rate, i.e., $t_{in}(1) = t$. For convenience, use $t(u, v) = \sum_{i \in [1, n-1]} route(u, i)route(v, i+1)t_{out}(i)$ to represent the traffic rate of f on link (u, v).

Consistent with most popular routing protocols, such as Cisco EIGRP and OSPF, our optimization objective is to minimize the path cost of flow f as shown in Equation (3), by calculating *route* and *place*. It should be noted that the proposed solutions can easily adapt to other optimization objectives, such as minimizing the maximum link load in the network.

$$\textbf{minimize} \sum_{i=1}^{n-1} \sum_{u \in V} \sum_{v \in V} route(u, i)route(v, i+1) \times weight(u, v, t_{out}(i) + load[u, v]) \quad (3)$$

subject to the following constraints:

$$\forall i > n, \forall v \in V : route(v, i) = 0 \quad (4)$$

$$route(src, 1) = 1, route(dst, n) = 1 \quad (5)$$

$$\forall v \in V : \sum_{i \in [1, n]} \sum_{m \in M} place(m, i)route(v, i) \leq sc[v] \quad (6)$$

$$\forall (u, v) \in E : t(u, v) + load[u, v] \leq bc[u, v] \quad (7)$$

$$\forall m \in M : \sum_{i \in [1, n]} place(m, i) = 1 \quad (8)$$

$$\forall m, m' \in M : \left(\sum_{i \in [1, n]} place(m', i)i - \sum_{i \in [1, n]} place(m, i)i \right) \times depend[m, m'] \geq 0 \quad (9)$$

$$\forall i \in [1,n] : t_{out}(i) = t_{in}(i) \prod_{m \in M} (1 + place_f(m,i)(ratio[m]-1)) \quad (10)$$

$$\forall i \in [1,n-1], \forall u,v \in V : route(u,i)route(v,i+1) \le connect[u,v] \quad (11)$$

Brief explanation of the model is as follows. Equation (3) defines the optimization objective. To discourage repeated links on the flow path, when the flow traverses the same link multiple times, the link weight of each traverse will be counted separately in the path cost. Equation (4) states that the n^{th} hop is the last hop of the flow path. Equation (5) enforces the first and last hops of the flow path to be the source src and destination dst, respectively. Equation (6) states that, for a node v, the total space demand of hosted middleboxes $\sum_{i\in[1,n]} \sum_{m\in M} place(m,i)route(v,i)$ should not exceed its space capacity $sc[v]$. Equation (7) states that, for a link (u,v), the aggregate load of all flows traversing it $t(u,v)+load[u,v]$ should not exceed its bandwidth capacity $bc[u,v]$. Equation (8) states that a middlebox m should be installed once and only once. Equation (9) enforces the dependency relation between middleboxes, or in other words m must be placed no later than m' if the former is depended on by the latter. Equation (10) states that, the outgoing flow traffic rate at a hop, i.e., $t_{out}(i)$, is equal to the incoming rate, i.e., $t_{in}(i)$, multiplying the traffic changing ratios $ratio[m]$ of all the middleboxes m placed at this hop, i.e., $\prod_{m\in M}(1 + place_f(m,i)(ratio[m]-1))$. It also ensures flow conservation, in the sense that no flow traffic can be generated or terminated at an intermediate node, except the effects of middleboxes. Equations (11) enforces that consecutive hops of the flow path must be connected in the network.

IV. Middlebox Placement with Predetermined Paths

In this section, we propose solutions for the TAPIM problem when the flow path, i.e., $route$, is predetermined. For example, in the tree topology, there is a unique path between any pair of leaves. We look at three cases of the problem. First, for the special case when there is no dependency between any middleboxes, we propose the Non-Ordered Set Placement algorithm that uses the least-first-greatest-last rule. Next, for the special case when there is a total dependency order on the middlebox set, we propose the dynamic programming based Totally-Ordered Set Placement algorithm. Finally, for the general scenario of a partial dependency order on the middlebox set, we prove that it is NP-hard by reduction from the Clique problem, and propose an efficient heuristic to convert a partially-ordered set to a totally-ordered set.

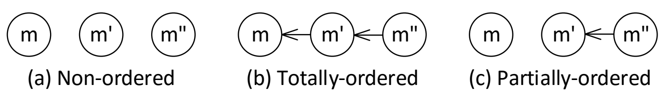


Fig. 2: Examples of non-ordered, totally-ordered, and partially-ordered middlebox sets.

A. Non-Ordered Middlebox Set

We start with the special case that the middlebox set M is a *non-ordered set*, i.e., $\forall m,m' \in M, m \not\leftarrow m'$ and $m' \not\leftarrow m$. Thus, different middleboxes can be placed in an arbitrary order. An example is shown in Fig. 2(a), where no dependency exists between middleboxes. We propose the *Non-Ordered Set Placement* (NOSP) algorithm, and show its optimality.

The basic idea is to shrink the flow as early as possible by installing the middleboxes that decrease the traffic rate from the path head, and expand the flow as late as possible by installing the middleboxes that increase the traffic rate from the path tail.

Apparently, the placement will succeed if the number of available spaces on the path is greater than or equal to the number of required middleboxes, i.e., $\sum_{i\in[1,n]} \sum_{v\in V} \left(route[v,i]sc[v] / \sum_{i'\in[1,n]} route[v,i'] \right) \ge |M|$. If there are enough spaces, the NOSP algorithm places the middleboxes as follows.

1) Sort all the middleboxes $m \in M$ based on their traffic changing ratios $ratio[m]$.
2) Place the middleboxes m with $ratio[m] < 1$ from the path head in an increasing order of their traffic changing ratios. When a node has no more space, continue with the succeeding node on the path.
3) Place the middleboxes m with $ratio[m] \ge 1$ from the path end in an decreasing order. When a node has no more space, continue with the proceeding node.

As can be seen, NOSP processes each middlebox in M only once after sorting, and therefore its time complexity is $O(|M| \log |M|)$, i.e., the time complexity to sort M.

Theorem 1. *The Non-Ordered Set Placement algorithm achieves the minimum path cost.*

The proof is omitted due to space limitations.

B. Totally-Ordered Middlebox Set

Next, we solve the other special case of TAPIM when the middlebox set is a totally-ordered set, i.e., $\forall m,m' \in M$, either $m \leftarrow m'$ or $m' \leftarrow m$, or in other words the middleboxes form a dependency chain. An example is shown in Fig. 2(b), in which m must be placed before m', and m' before m''. Although the placement order of the middleboxes has been determined, it is still necessary to determine the optimal placement location for each middlebox, because there may be an excessive number of available spaces on the flow path. For easy description, we use m_j to denote the j^{th} middlebox from the head of the dependency chain, and v_i to denote the i^{th} hop node on the flow path, where i and j start from 1.

We propose a dynamic programming based algorithm called *Totally-Ordered Set Placement* (TOSP) based on the following observation. Use TOSP(i,j) to denote the minimum weight sum of the first i links when placing the first j middleboxes, i.e., m_1 to m_j, on the first i hops, i.e., v_1 to v_i, of the flow path. The optimal substructure gives the following recursive formula.

$$\text{TOSP}(i,j)=\begin{cases} w(1;1,j), \text{ if } i=1. \\ \min_{x\in[1,j+1]}\Big(\text{TOSP}(i-1,x-1)+ \\ \quad w(i;x,j)\Big), \text{ otherwise.}\end{cases} \tag{12}$$

where $w(i;x,j)$ is the weight of link (v_i, v_{i+1}) when placing middleboxes m_x to m_j on node v_i, i.e.,

$$w(i;x,j)=\begin{cases} weight\Big(v_i, v_{i+1}, t\prod_{y\in[1,j]} ratio[m_y]+ \\ \quad load[v_i,v_{i+1}]\Big), \text{ if } sc[v_i]\geq j-x+1. \\ \infty, \text{ otherwise.}\end{cases} \tag{13}$$

Equation (12) states that if $i = 1$, TOSP(i, j) is simply the weight of the first path link when placing the first j middleboxes on the first hop v_1. Otherwise, the optimal result TOSP(i, j) to place the first j middleboxes on the first i hops is to select the minimum link weight sum among $j + 1$ possible solutions, in which the x^{th} solution places the first $x - 1$ middleboxes on the first $i - 1$ hops, i.e., TOSP$(i - 1, x - 1)$, and places the remaining middleboxes m_x to m_j on the i^{th} hop v_i, i.e., $w(i; x, j)$.

Equation (13) calculates the weight of the i^{th} path link, i.e., (v_i, v_{i+1}), when placing middleboxes m_x to m_j on the i^{th} hop v_i, and sets it to infinity if v_i has fewer than $j-x+1$ available spaces. Note that if $x \leq j$ and v_i has sufficient spaces, $w(i; x, j)$ does not depend on x. In other words, as long as v_i has no fewer than $j-x+1$ spaces to host middleboxes m_x to m_j, the weight of link (v_i, v_{i+1}) is the same, which simplifies the calculation of TOSP(i, j) as the sum of the minimum sub-solution $\min_{x\in[j-sc[v_i]+1,j+1]}\big\{\text{TOSP}(i-1,x-1)\big\}$ and a constant.

When there is no repeating node on the flow path, the time complexity of the TOSP algorithm is $O(n|M|^2)$, because the dynamic programming table has n rows and $|M|$ columns, and it takes up to $O(|M|)$ time to calculate each table entry.

When the flow path $route$ is not efficient and contains repeating nodes, the above algorithm may obtain a sub-optimal result. The reason is that, different hops of a repeating node share middlebox spaces, but the above algorithm processes those hops always from the path head, and thus assigns earlier hops higher priority. A simple solution is to first enumerate all the possibilities to divide the shared spaces among different hops of a repeating node, and then apply TOSP to each possible division. For example, if the flow path contains a repeating node with s spaces that appears twice at the i_1^{th} hop v_{i_1} and the i_2^{th} hop v_{i_2}, we view v_{i_1} and v_{i_2} as two independent nodes by allocating $x \in [0, s]$ spaces to v_{i_1} and $s - x$ spaces to v_{i_2}. TOSP is then applied to each different x value, and the minimum path cost among all the cases is the optimal solution.

C. Partially-Ordered Middlebox Set

We now solve the general scenario where the dependency relation is a partial order. The following theorem shows the NP-hardness of the problem.

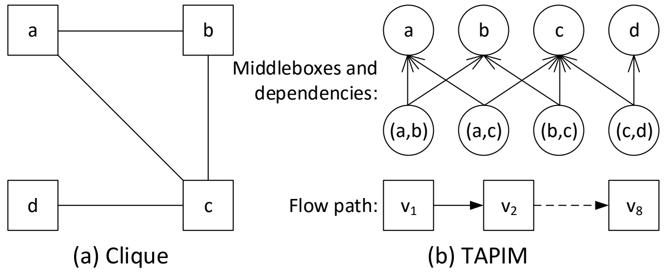


Fig. 3: Reduction from Clique to TAPIM with predetermined path.

Theorem 2. *The TAPIM problem with a predetermined path for a partially ordered middlebox set is NP-hard.*

Proof. We prove by reduction from the Clique problem [12]. The clique problem decides whether an undirected graph $G = (V, E)$ has a clique of size k, which is a complete sub-graph with k vertices and $\binom{k}{2}$ edges. For example, the graph in Fig. 3(a) has a clique of size 3: $(\{a, b, c\}, \{(a, b), (a, c), (b, c)\})$.

Given an instance of the Clique problem with a graph $G = (V, E)$, an instance of the TAPIM problem can be constructed in polynomial time as follows.

1) For each vertex $p \in V$, create a vertex middlebox m_p with $ratio[m_p] = 2$.
2) For each edge $(p, q) \in E$, create an edge middlebox $m_{(p,q)}$ with $ratio[m_{(p,q)}] = 2^{-k/\binom{k}{2}}$.
3) The middlebox corresponding to an edge (p, q) depends on the two middleboxes corresponding to its two incident vertices p and q, i.e., $m_p \leftarrow m_{(p,q)}$ and $m_q \leftarrow m_{(p,q)}$.
4) There is a single flow f with the initial traffic rate of one, i.e., $t = 1$. The path has $|V| + |E|$ nodes. Use v_i to denote the i^{th} node on the path. Each node has a space capacity of one, i.e., $sc[v_i] = 1$.
5) Each link on the path has a bandwidth capacity of infinity, i.e., $bc[v_i, v_{i+1}] = \infty$. The link $(v_{k+\binom{k}{2}}, v_{k+\binom{k}{2}+1})$ is called the critical link, with its weight being one if the link load is no more than one and infinity otherwise, i.e.,

$$weight((v_{k+\binom{k}{2}}, v_{k+\binom{k}{2}+1}), l)=\begin{cases}1, \text{ if } l\leq 1. \\ \infty, \text{ if } l>1.\end{cases} \tag{14}$$

The weight of any other link is always zero.

Next, it can be shown that if the graph $G = (V, E)$ has a clique of size k, then the constructed TAPIM instance has a minimum path weight of one, and vice versa. The proof detail is omitted due to space limitations. □

After proving the NP-hardness, our solution to place a partially-ordered middelbox set is to first convert it to a totally-ordered middlebox set and then apply TOSP.

Following the least-first-greatest-last rule in NOSP, the objective of the conversion algorithm is to arrange the middleboxes in the resulting total order chain in the increasing order of their traffic changing ratios. The intuitive solution is thus to iteratively find the middleboxes without dependence, remove among them the one with the least traffic changing ratio, and add it to the end of the total order chain. For example, given four middleboxes with the following traffic changing ratios and dependencies: $1.4 \leftarrow 1.5$ and $1.6 \leftarrow 0.1$, the conversion result will be the following total order chain: $1.4 \leftarrow 1.5 \leftarrow 1.6 \leftarrow 0.1$.

To increase the solution search space, we also add lookahead information by searching further beyond just the middleboxes without dependence. Define a self-dependent middlebox tree of size k to be a tree of k middleboxes that are rooted from a single middlebox and depend on only the middleboxes in the tree. The traffic changing ratio of the tree is the product of the traffic changing ratios of all the middleboxes in the tree. In the above example, $1.6 \leftarrow 0.1$ is a self-dependent tree of size 2, and its traffic changing ratio is $1.6 \times 0.1 = 0.16$.

The conversion algorithm with a lookahead value of k works in iterations as follows. In each iteration, the algorithm first finds all the middleboxes with no dependence. Using each of such middleboxes as the root, the algorithm calculates the self-dependent tree of size up to k that has the minimum traffic changing ratio. Among all the calculated trees with different root middleboxes, the algorithm selects the one with the minimum traffic changing ratio, removes its root middlebox, and adds the middlebox to the total order chain. For the above example, the first iteration generates two trees of size up to 2: 1.4 of size 1 with 1.4 being the root, and $1.6 \leftarrow 0.1$ of size 2 with 1.6 being the root. Since the traffic changing ratio of the latter 0.16 is less than that of the former 1.4, the root of the latter will be removed. The resulting total order chain after the algorithm converges is thus: $1.6 \leftarrow 0.1 \leftarrow 1.4 \leftarrow 1.5$.

When the lookahead parameter $k = 1$ or 2, the time complexity of the conversion algorithm is $O(|M| \log |M|))$, because there are up to $|M|$ iterations, and the time complexity to select the middelebox with the minimum traffic changing ratio is $O(\log |M|)$ using a heap. When $k = 2$, the optimal self-dependent trees of size up to 2 with each middlebox being the root can be pre-calculated in $O(|M|)$ time. Since $|M|$ is usually small, $k = 2$ will be sufficient in most cases.

V. Middlebox Placement without Predetermined Path

In this section, we solve the TAPIM problem when the flow path, i.e., *route*, is not predetermined. We start by proving that the TAPIM problem without a predetermined path is NP-hard even for a non-ordered set by reduction from the Hamiltonian Cycle problem. We then propose a two-step solution by first finding a flow path with sufficient spaces and then applying the algorithms in Section IV to place middleboxes on the given path.

A. NP-Hardness

The following theorem shows the hardness of the TAPIM problem without a predetermined flow path.

Theorem 3. *Without a predetermined flow path, the TAPIM problem is NP-hard even for a non-ordered middlebox set.*

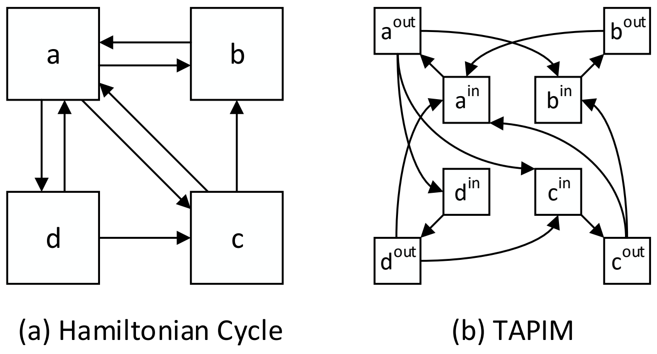


Fig. 4: Reduction from Hamiltonian Cycle to TAPIM.

Proof. The proof is by reduction from the Hamiltonian Cycle problem, which determines for a directed graph $G = (V, E)$ whether there exists a simple cycle that contains each vertex in V. Note that a Hamiltonian cycle must be a simple cycle without repeating nodes.

For a Hamiltonian Cycle instance with a graph $G = (V, E)$, a TAPIM instance with a graph $G' = (V', E')$ can be constructed in polynomial time as follows:

1) For each vertex $v \in V$, create two nodes $v^{in}, v^{out} \in V'$, where v^{in} has a space capacity of zero, i.e., $sc[v^{in}] = 0$, and v^{out} of one, i.e., $sc[v^{out}] = 1$. Connect the two nodes with a link $(v^{in}, v^{out}) \in E'$, and set its bandwidth capacity to infinity, i.e., $bc[v^{in}, v^{out}] = \infty$. Its weight is one if the load is no more than one, i.e., $\forall l \leq 1, weight(v^{in}, v^{out}, l) = 1$, and infinity otherwise.
2) For each edge $(u, v) \in E$, create a link $(u^{out}, v^{in}) \in E'$, and set its bandwidth capacity to one, i.e, $bc[u^{out}, v^{in}] = 1$, and its weight to be zero, i.e., $\forall l, weight(u^{out}, v^{in}, l) = 0$. An example to create G' from G is shown in Fig. 4.
3) Create a flow f with the source and destination both being a^{in}, i.e., $src = dst = a^{in}$, where a is an arbitrary vertex in V. The initial traffic rate is one, i.e., $t = 1$. The middlebox set M is a non-ordered one with $|V|$ middleboxes, i.e., $|M| = |V|$, and each middlebox has a traffic changing ratio of one, i.e., $\forall m \in M, ratio[m] = 1$.

Next, it can be shown that if G has a Hamiltonian cycle, then the TAPIM instance with G' and f has a minimum path cost of $|V|$, and vice versa. The proof detail is omitted due to space limitations. □

B. Traffic and Space Aware Routing

Our solution to TAPIM without a predetermined path works in two steps by first finding a viable path for the flow and then applying the algorithms in Section IV to place the middleboxes on the determined path.

From the proof of Theorem 3, it can be seen that it is NP-hard to find the minimum cost path with sufficient spaces, and thus we propose the Traffic And Space Aware Routing (TASAR) heuristic. The basic idea is to originate from the source, iteratively route to a nearby node with spaces until sufficient spaces have been accumulated, and finally go to the destination.

In detail, the TASAR heuristic works as follows. It starts by calculating the number of spaces needed on the path besides those in the source and destination, i.e., $|M| - sc[src] - sc[dst]$. Next, the heuristic enters iterative loops to accumulate the necessary number of spaces. In the x^{th} iteration, the heuristic runs Dijkstra's algorithm [12] from v_x, with $v_1 = src$, to find the nearest (in terms of the path cost) node with spaces, denoted as v_{x+1}, and add the path from v_x to v_{x+1} to the flow path $route$. If sufficient spaces have been accumulated, i.e., $|M| - sc[src] - sc[dst] - \sum_{i=1}^{x+1} sc[v_i] \leq 0$, the iteration stops, and the heuristic runs Dijkstra's algorithm for the last time to find the minimum cost path from the current node v_{x+1} to the destination dst. Otherwise, if more spaces are needed, the iteration continues.

The time complexity of the heuristic is $O(|M|(|E| + |V| \log |V|))$, because there will be up to $O(|M|)$ iterations, and the time complexity of each iteration is that of Dijkstra's algorithm $O(|E| + |V| \log |V|)$.

VI. Experiment and Simulation Results

We use a combination of simulations and experiments for performance evaluation. We have conducted simulations to obtain performance data in large scale networks, and have also built a prototype to validate the solutions in realistic environments. In this section, we present extensive simulation and experiment results to show the effectiveness of our design.

A. Simulation Results

We have implemented the proposed algorithms in the NS-3 simulator, and used the following performance metrics for benchmark comparisons.

1) *End-to-end delay:* The end-to-end delay is the interval to transmit a packet from its source to destination.
2) *Packet loss ratio:* The packet loss ratio is the percentage of packets lost with respect to packets sent.

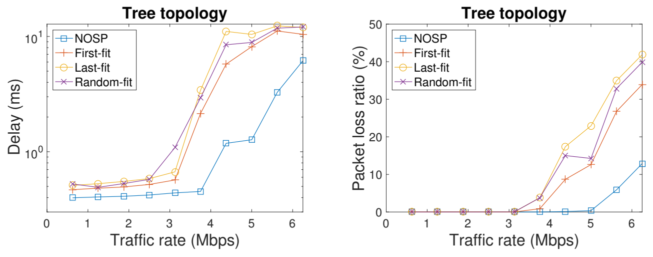
Heavier congestion will cause longer end-to-end delays and higher packet loss ratios.

To reflect the burstiness of realistic traffic, we adopt the on-off traffic model. When a flow f is in the on state, its initial traffic rate t is the product of a baseline rate and a random number between 0.5 to 1.5; when in the off state, its traffic rate is zero. A flow is in each of the two states for 50% of the time. There are two candidate middlebox sets with different traffic changing ratios and dependency relations, and each flow will randomly choose one of them. Each link in the network has a bandwidth capacity of 100 Mbps and a propagation delay of 2 μs. We use the Cisco EIGRP link weight function [2] by setting only $K2$ to one. Every simulation run lasts five minutes, and the presented result is the average of four simulation runs.

1) Placing Non-Ordered Set with Predetermined Path: For the NOSP algorithm to place a non-ordered middlebox set on a predetermined path, since there are no existing solutions for the studied problem, we designed the following three benchmark algorithms. Same as NOSP, all the benchmark algorithms sort the middleboxes based on their traffic changing ratios before placing them.

1) *First-fit* continuously places the sorted middleboxes in the increasing order from the head of the flow path.
2) *Last-fit* continuously places the sorted middleboxex in the decreasing order from the tail of the flow path.
3) *Random-fit* randomly places the sorted middleboxes on random nodes on the path that have spaces.

We pick the tree topology, since it is a popular choice among institutional networks, and there is only a single path between any pair of nodes. We set up a four-layer quad-tree with 21 switches and 64 hosts. Each switch has 13 spaces to ensure sufficient spaces for all flows. Each host generates a flow to a random destination. The two candidate sets of middleboxes are: {0.7, 0.8, 1.1, 1.2} and {0.8, 0.9, 1.1, 1.3}. The baseline traffic rate of each flow ranges from 0.625 to 6.25 Mbps with a stride of 0.625 Mbps.



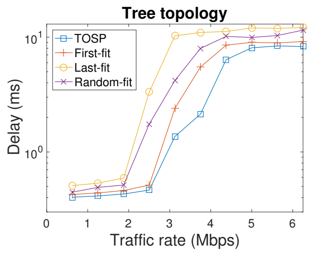
(a) End-to-end delay. (b) Packet loss ratio.

Fig. 5: NOSP simulation results.

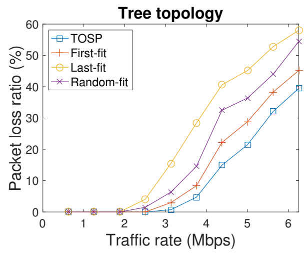
Fig. 5(a) shows the average end-to-end delays of the four algorithms. We can see that NOSP consistently achieves the shortest delay due to its optimal middlebox placement scheme. On the other hand, Last-fit has the worst performance, because it places middleboxes at the path tail, and the flow rate is 1× on most links of the path. By contrast, First-fit achieves a relatively shorter average delay by placing middleboxes at the path head. The reason is that half of the flows picked the first set of middleboxes with an aggregate traffic changing ratio of $0.7 \times 0.8 \times 1.1 \times 1.2 = 0.74$, and the other half picked the second set with a ratio of $0.8 \times 0.9 \times 1.1 \times 1.3 = 1.03$, so on average the middleboxes placed at the path head would reduce the traffic rate of a flow to $(0.74 + 1.03)/2 = 0.885\times$, which is the traffic rate on most links of the path. Finally, the delay of Random-fit is between that of Last-fit and First-fit due to its randomized strategy.

Fig. 5(b) plots the packet loss ratios. We can observe a similar trend that NOSP always achieves the lowest packet loss ratio. When the flow traffic rate is small, all the algorithms have zero packet loss ratios. Compared with NOSP, other algorithms experience packet loss at much smaller traffic rates, and their ratios increase much faster.

2) Placing Totally-Ordered Set with Predetermined Path: Next, we evaluate the TOSP algorithm to place a totally-ordered set with similar benchmark algorithms as above, in which First-fit, Last-fit, and Random-fit place the middleboxes following the given total order chain from the path head, tail, and randomly, respectively. The traffic changing ratios and dependency chains of the two candidate sets of middleboxes are: $\{0.8 \leftarrow 1.1 \leftarrow 0.7 \leftarrow 1.2\}$ and $\{1.2 \leftarrow 0.7 \leftarrow 1.1 \leftarrow 0.8\}$. Other simulation settings are the same as above.



(a) End-to-end delay.

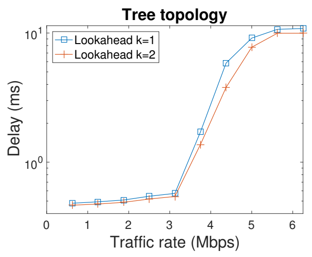


(b) Packet loss ratio.

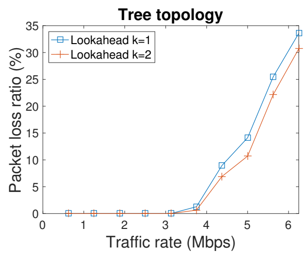
Fig. 6: TOSP simulation results.

As shown in Fig. 6(a), TOSP achieves the shortest end-to-end delay because of its dynamic programming based optimal middlebox placement scheme. Similar as above, the delays of the other three algorithms increase in the sequence of First-fit, Random-fit, and Last-fit. The packet loss ratio results in Fig. 6(b) are consistent, and TOSP consistently outperforms others.

3) Placing Partially-Ordered Set with Predetermined Path: To evaluate the placement of partially-ordered middlebox sets on predetermined paths, we use the proposed heuristic to convert the partially-ordered sets to fully-ordered sets, and then apply TOSP. We adjust the lookahead parameter k from one to two and compare their performances. The traffic changing ratios and the dependencies of the two candidate sets of middleboxes are: $\{1.1 \leftarrow 0.8, 1.2 \leftarrow 0.7\}$ and $\{1.1 \leftarrow 1.2, 1.3 \leftarrow 0.7\}$. Note that different total order chains will be generated when using the two different lookahead values.



(a) End-to-end delay.



(b) Packet loss ratio.

Fig. 7: Partial to total order conversion simulation results.

Fig. 7(a) compares the end-to-end delays of the two different lookahead values. We can see that the lookahead value of two achieves shorter delays with a deeper search into the solution space. Similarly, Fig.7(b) shows that the lookahead value of two achieves a lower average packet loss ratio.

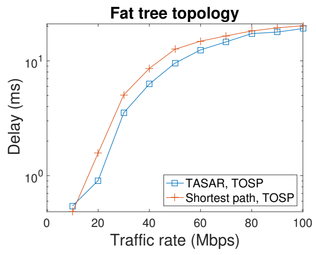
4) Placing Middleboxes without Predetermined Path: Finally, we evaluate the Traffic And Space Aware Routing (TASAR) heuristic by comparing it with a hop count based and ECMP (i.e., load-balancing) enabled shortest path routing algorithm. For the multi-path topology, we choose an 8-pod fat tree with 80 switches and 128 hosts. Each host generates a flow to a random host in other pods. The baseline traffic rate of each flow ranges from 10 to 100 Mbps with a stride of 10 Mbps. The two sets of candidate middleboxes after conversion are: $\{1.2 \leftarrow 0.7 \leftarrow 1.1 \leftarrow 0.8\}$ and $\{1.3 \leftarrow 0.7 \leftarrow 1.1 \leftarrow 1.2\}$.

We first compare the routing success ratios of the two algorithms. We adjust the number of spaces per switch from 6 to 8, and measure the percentage of flows that can successfully find paths with sufficient middlebox spaces. As shown in Table I, when the space number per switch is 6, the routing success ratio of TASAR is 5% higher than that of shortest path routing. When the number increases 7, TASAR achieves a 100% routing success ratio, while shortest path routing cannot find path for 3.75% of the flows. Finally, when it increase to 8, both algorithms achieve 100% routing success ratios.

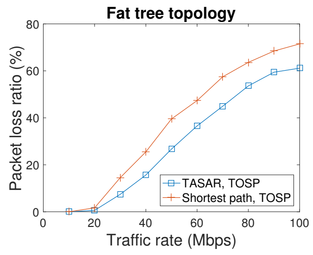
Spaces per switch	TASAR	Shortest path routing
6	93.75%	88.91%
7	100%	96.25%
8	100%	100%

TABLE I: Flow routing success ratio.

Next, we fix the space number per switch to 8, and compare the end-to-end delays and packet loss ratios of the two algorithms. As shown in Fig. 8(a), when the baseline flow rate is 10 Mbps, TASAR has a slight longer delay, because its paths are not as short as those generated by shortest path routing. However, once the flow rate increases beyond 10 Mbps, TASAR consistently delivers shorter delays due to its traffic awareness in path calculation. Fig. 8(b) also shows that TASAR consistently achieves lower packet loss ratios.



(a) End-to-end delay.



(b) Packet loss ratio.

Fig. 8: TASAR simulation results.

B. Experiment Results with Prototype

We have implemented an SDN based prototype using the open-source SDN controller Floodlight and emulation platform Mininet, and developed a middlebox emulator using the libpcap library [8]. Due to space limitations, the detailed description of the implementation is omitted. We validate our design by comparing TOSP with TASAR routing and TOSP with shortest path routing.

We pick the Abilene backbone topology [1], as shown in Fig. 9. Each node has a space capacity of three, and each link

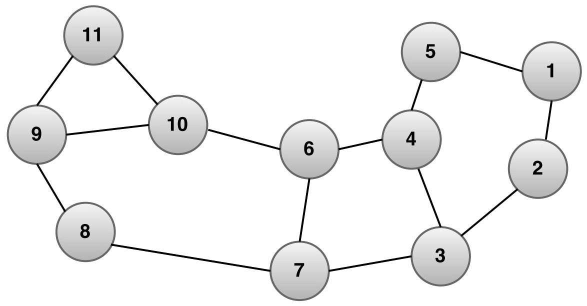
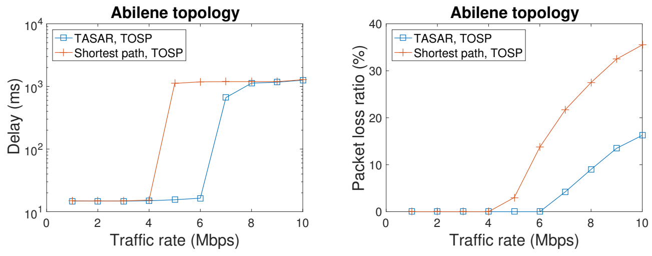


Fig. 9: Abilene backbone network topology.

has a bandwidth capacity of 10 Mbps. For traffic generation, we use Iperf to create constant bit rate UDP traffic flows. Four flows are generated: two from node 1 to 8, and two from 11 to 2. The initial traffic rate of each flow ranges from 1 to 10 Mbps with a stride of 1 Mbps. Each flow needs four middleboxes with the following traffic changing ratios and total order chain after conversion: $\{1.2 \leftarrow 0.7 \leftarrow 1.1 \leftarrow 0.8\}$.



(a) End-to-end delay. (b) Packet loss ratio.

Fig. 10: Prototype experiment results.

As can be seen in Fig. 10(a), the experiment data are consistent with the simulation results, and TASAR achieves shorter end-to-end delays than shortest path routing. Also, Fig. 10(b) shows that the former achieves much lower packet loss ratios.

VII. Conclusions

The advancement of virtualization technology has made NFV a promising platform for network function provisioning. However, the flexibility to run an NFV middlebox on any available standard server also creates a challenge for efficient NFV implementation. In this paper, we have studied the optimal placement of NFV middleboxes by considering different middlebox traffic changing effects and dependency relations. We first formulate the Traffic Aware Placement of Interdependent Middleboxes problem as a graph optimization problem with the objective to load-balance the network. Next, we solve the problem when the flow path is predetermined, and propose optimal algorithms for a non-ordered or totally-ordered middlebox set. For the general scenario of a partially-ordered middlebox set, we show that the problem is NP-hard by reduction from the Clique problem, and propose an efficient heuristic to convert a partially-ordered set to a totally-ordered one. On the other hand, when the flow path is not predetermined, we prove that the studied problem is NP-hard even for a non-ordered middlebox set by reduction from the Hamiltonian Cycle problem, and propose the Traffic And Space Aware Routing heuristic. We have conducted large scale simulations to evaluate the proposed solutions, and have also implemented an SDN based prototype to validate them in realistic environments. Extensive simulation and experiment results are presented to show the effectiveness of our design.

References

[1] "Abilene backbone," http://abilene.internet2.edu.
[2] "Cisco EIGRP," http://www.cisco.com/c/en/us/support/docs/ip/enhanced-interior-gateway-routing-protocol-eigrp/16406-eigrp-toc.html.
[3] "Cisco: NAT order of operation," http://www.cisco.com/c/en/us/support/docs/ip/network-address-translation-nat/6209-5.html.
[4] "Citrix CloudBridge," https://www.citrix.com/content/dam/citrix/en_us/documents/products-solutions/cloudbridge-technical-overview.pdf.
[5] "Microsoft TechNet: VPNs and firewalls," https://technet.microsoft.com/en-us/library/cc958037.aspx.
[6] "Network functions virtualisation white paper #3," https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper3.pdf.
[7] "OSPF: frequently asked questions," http://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/9237-9.html.
[8] "TCPDUMP/LIBPCAP public repository," http://www.tcpdump.org/.
[9] A. Curtis *et al.*, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *IEEE INFOCOM*, 2011.
[10] A. Singh *et al.*, "Server-storage virtualization: integration and load balancing in data centers," in *ACM/IEEE Supercomputing*, 2008.
[11] B. Heller *et al.*, "Elastictree: saving energy in data center networks," in *USENIX NSDI*, 2010.
[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.
[13] W. Ding *et al.*, "OpenSCaaS: an open service chain as a service platform toward the integration of SDN and NFV," *IEEE Network*, vol. 29, no. 3, pp. 30–35, 2015.
[14] J. Anderson *et al.*, "xOMB: extensible open middleboxes with commodity servers," in *ACM/IEEE ANCS*, 2012.
[15] J. Hwang *et al.*, "NetVM: high performance and flexible networking using virtualization on commodity platforms," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 34–47, 2015.
[16] J. Martins *et al.*, "ClickOS and the art of network function virtualization," in *USENIX NSDI*, 2014.
[17] J. Matias *et al.*, "Toward an SDN-enabled NFV architecture," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 187–193, 2015.
[18] H. Jin *et al.*, "Efficient VM placement with multiple deterministic and stochastic resources in data centers," in *IEEE GLOBECOM*, 2012.
[19] Y. Li, L. Phan, and B. T. Loo, "Network functions virtualization with soft real-time guarantees," in *IEEE INFOCOM*, 2016.
[20] W. Ma *et al.*, "Traffic-aware placement of NFV middleboxes," in *IEEE GLOBECOM*, 2015.
[21] M. J. Miller, B. Vucetic, and L. Berry, *Satellite communications: mobile and fixed services*. Springer Science & Business Media, 1993.
[22] H. Moens and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *IEEE CNSM*, 2014.
[23] T. Mori *et al.*, "Identifying elephant flows through periodically sampled packets," in *ACM IMC*, 2004.
[24] R. Cohen *et al.*, "Near optimal placement of virtual network functions," in *IEEE INFOCOM*, 2015.
[25] S. Fayazbakhsh *et al.*, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *USENIX NSDI*, 2014.
[26] S. Mehraghdam *et al.*, "Specifying and placing chains of virtual network functions," in *IEEE Cloud Networking*, 2014.
[27] V. Sekar *et al.*, "Design and implementation of a consolidated middlebox architecture," in *USENIX NSDI*, 2012.
[28] Z. Qazi *et al.*, "SIMPLE-fying middlebox policy enforcement using SDN," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 27–38.