2016 IEEE 41st Conference on Local Computer Networks

Throughput Maximization in Software-Defined Networks with Consolidated Middleboxes

Meitian Huang†, Weifa Liang†, Zichuan Xu‡†, Mike Jia†, and Song Guo¶

† The Australian National University, Canberra, ACT 2601, Australia
‡ University College London, London WC1E 7JE, UK
¶ The Hong Kong Polytechnic University, Hong Kong
Email: u4700480@anu.edu.au, wliang@cs.anu.edu.au, z.xu@ucl.ac.uk, u5515287@anu.edu.au, cssongguo@comp.polyu.edu.hk

***Abstract*—Today's computer networks rely on a wide spectrum of specialized middleboxes to improve their security and performance. Traditional middleboxes that are implemented by dedicated hardware are expensive and hard to manage. A promising technique of consolidated middleboxes – implementing traditional middleboxes in Virtual Machines (VMs) – offers economical yet simplified management of middleboxes in Software-Defined Networks (SDNs). However there are still challenges to realizing user routing requests with network function enforcement (a sequence of middleboxes) while maximizing the network throughput, due to various resource constraints on SDNs, such as forwarding table capacity at each switch, bandwidth resource capacity at each link, and computing resource capacity at each server (Physical Machine). In this paper, we study the problem of maximizing the network throughput of an SDN by admitting as many user requests as possible, where each user request has both bandwidth and computing resource demands to implement its network functions (consolidated middleboxes). We first formulate the problem as a novel network throughput maximization problem. We then provide an Integer Linear Program (ILP) solution for it if the problem size is small, otherwise, we devise two heuristics that strive for the fine tradeoff between the accuracy of solutions and the running times of achieving the solutions. We finally evaluate the performance of the proposed algorithms by simulations, based on real and synthetic network topologies. Experimental results demonstrate that the proposed algorithms are very promising.**

***Keywords*—software-defined networking, network function virtualization, consolidated middleboxes, routing algorithms, network resource allocation.**

I. Introduction

Computer networks nowadays rely on various middleboxes, including firewall, Intrusion Detection Systems (IDSs), WAN optimizers, and Deep Packet Inspection (DPI), to enhance the performance and security of different network services [7], [11], [20]. Unfortunately, the management and deployment of hardware middleboxes are highly complex and costly [20]. With the advancement of the Network Function Virtualization (NFV), middleboxes can be implemented in Virtual Machines (VMs) that run in Physical Machines (PMs) [6], [18], [20]. We refer to the software implementations of middleboxes as the *consolidated middleboxes*. Along with the technique of Software-Defined Networking (SDN), consolidated middleboxes offer a promising alternative method to provide cheap and simplified management of middleboxes [9], [19].

In this paper we deal with realizing user requests with each specifying a sequence of middleboxes in an SDN with the aim to maximize the throughput of the network. This problem poses great challenges: one is that there are many different types of resources in an SDN with limited capacities. For instance, the forwarding table of an SDN-enabled switch usually is made by Ternary Content-Addressable Memory (TCAM) to facilitate fast, parallel lookups of forwarding rules. However, TCAM is expensive and energy hungry, its capacity thus is restricted to a few thousands [13]. Meanwhile, the computing resource of the PM attached to an SDN-enabled switch is limited too. Another challenge is that the resources in an SDN are allocated dynamically, causing significant fluctuations in their consumptions and availabilities. Such time-varying nature of resource demands and consumptions complicates the cost modeling of resource usages. In addition, each user request requires its traffic to traverse a specified sequence of middleboxes that is referred to the *service chain* of the request. To tackle the challenges, in this paper we innovatively propose a cost model to accurately capture dynamic resource consumptions in an SDN. We then propose efficient algorithms that jointly meet the service chains and resource demands on traffic routing of various user requests. Despite that there are several studies of consolidated middleboxes [3], [9], [18], none of them has taken the forwarding table size into consideration, and they provided suboptimal solutions to the problem by decomposing routing from service chain execution [9]. To the best of our knowledge, we are the first to formulate a novel routing optimization problem in SDNs with consolidated middleboxes that incorporates various resource capacity constraints and different user QoSs, by providing efficient heuristic solutions.

The main contributions of this paper are summarized as follows. We consider the network throughput maximization problem in SDNs, subject to various capacity constraints of network resources and user resource demand constraints. We first formulate an Integer Linear Program solution to the problem when the problem size is small. We then devise a heuristic for the problem through introducing a novel cost modeling of resource consumptions and problem reduction.

DOI 10.1109/LCN.2016.58

To respond to user requests quickly, we also propose a faster heuristic by exploring the finest tradeoff between the accuracy of the solution obtained and the running time of finding such a solution. We finally evaluate the performance of the proposed algorithms by simulations, based on real and synthetic network topologies and using synthetic traffic traces. Experimental results demonstrate that the proposed algorithms are very promising.

The rest of the paper is organized as follows. Section II will review related work. Section III will introduce the system model and notations, and define the problem. Section IV will formulate an ILP solution to the problem. Sections V and VI will present a heuristic algorithm and a faster heuristic algorithm, respectively. Section VII will evaluate the performance of the proposed algorithms with simulations.

II. Related Work

While middleboxes are widely used to guarantee security and performance of routing traffic in contemporary computer networks, the deployment of traditional hardware middleboxes incurs high capital investment [19] and high operational costs [20]. To tackle these issues, recent efforts on new frameworks and architectures of consolidated middleboxes [6], [8], [16], [19], have resulted in promising alternatives to traditional hardware middleboxes. For example, Sekar *et al.* devised an architecture CoMb [19] that focused on consolidating software-based implementations of middlebox functions on a shared hardware platform. Qazi *et al.* developed SIMPLE [18] that enforces high-level routing policies for middlebox-specific traffic steering based on SDN. One fundamental problem that has not been addressed by existing studies is network throughput maximization while meeting various resource constraints and user QoSs. A few recent studies investigated this issue [3], [9], however they neither considered resource constraints such as the forwarding table size constraint on switches nor took global optimization approaches, thereby the solutions delivered are suboptimal, e.g., Charikar *et al.* [3] assumed that every switch in a network can perform middlebox functions without taking forwarding table sizes into consideration. Gushchin *et al.* [9] assumed that the routing traffic of a request can be split into multiple paths, and proposed a two-stage local optimization. In [14], the authors studied a problem of VM placement and path selection, striking for a tradeoff between link and server usage. This work, however, is different from ours because they assumed that multiple requests of the network function can be satisfied using a single VM that implements the network function. On the other hand, Li *et al.* presented the design and implementation of a system that dynamically provisions resources to provide timing guarantees with the objective of the number of request admitted to the cloud, while meeting the deadline requirements of admitted requests [15].

III. Preliminaries

A. System Model

We consider a software-defined network represented by a directed graph $G = (V, E)$, where V is the node set and E is the edge set. Each node $v \in V$ represents an SDN-enabled switch, while each directed edge $\langle u, v\rangle \in E$ represents an Internet link from switch u to switch v. Each switch $v \in V$ is equipped with a Ternary Content-Addressable Memory (TCAM) forwarding table that can accommodate at most L_v forwarding rules. A subset of switches in V are connected to physical machines (PMs) to implement middleboxes as virtual machines. As each such switch and its attached PM are usually connected by a high-speed optical link, the latency between them is negligible and the switches and their attached PMs will be used interchangeably. Denote by V_{pm} ($\subseteq V$) the set of switches that have attached PMs. We assume that each PM attached to a switch $v \in V_{pm}$ has limited capacity resource, and denote by C_v its computing resource capacity. If switch $v \in V \setminus V_{pm}$, then $C_v = 0$. Similarly, each link $e \in E$ has a bandwidth capacity B_e. We assume that there is a centralized SDN controller for network G that collects and processes user requests by installing forwarding rules into the forwarding tables in switches, assigning the middleboxes for the requests to PMs, and allocating bandwidth on links.

B. User Requests

We assume that time is slotted into *equal time slots*. User requests are scheduled by the centralized SDN controller in the beginning of each time slot. Let $S(t)$ be the set of arrived user requests in time slot t. Each *user request* has a certain amount of bandwidth demand to route its traffic in G from a source switch to a destination switch that passes through a sequence of middleboxes, and the request also has an end-to-end delay requirement. Let $r_i \in S(t)$ be a user request, represented by a quintuple $r_i = \langle s_i, t_i, b_i, SC_i, d_i\rangle$, where $s_i, t_i \in V$ are, respectively, its source and destination switches, b_i is its bandwidth demand, SC_i is its service chain, and $d_i \in \mathbb{R}^+$ is its end-to-end delay constraint. Admission of user request r_i therefore involves routing the traffic from the source switch s_i to the destination switch t_i via a routing path $P_i = \langle s_i, \ldots, t_i\rangle$ subjects to constraints b_i, SC_i, and d_i. Service chain SC_i is a sequence of services that are chained together and has to be traversed in the specified order by the traffic of r_i.

Following the same assumption as in [9], [16], [18], [19], we assume that services in SC_i are run in a single VM and different VMs serving different requests can be consolidated to a single Physical Machine (PM). Specifically, when the traffic of request r_i arrives at the PM hosting the VM for its service chain SC_i, it will be directed to the VM and the services in SC_i are applied in the specified order. Performing the services in SC_i for r_i thus will consume the computing resource of a PM. Denote by $C(i, j)$ the amount of computing resource needed by SC_i in a PM attached to the switch $v_j \in V_{pm}$. Notice that some services in SC_i may alter the volume of the traffic of request r_i. For instance, the volume of traffic increases if encryption is applied, while the volume of traffic decreases if compression is applied. We here define $\lambda_i \in \mathbb{R}^+$ as the ratio between the volumes of the traffic of request r_i after and before processing. Since request r_i requires an amount

b_i of bandwidth to route its traffic before processing, it thus needs an amount $\lambda_i \cdot b_i$ of bandwidth to route the processed traffic. The value of λ_i for each request r_i is given and can be derived from historical traffic [4]. In addition, each request r_i has a tolerant end-to-end delay requirement d_i. Suppose r_i is admitted with a routing path P_i from its source s_i to its destination t_i, and its service chain SC_i is implemented on a PM-attached switch $v \in V_{pm}$ on P_i. Let $D_n(P_i)$ and $D_p(i, v)$ be the network delay experienced by r_i via path P_i and the processing delay of r_i at PM v, respectively. The network delay $D_n(P_i)$ is proportional to the number of switches on P_i, and the average processing delay $D_p(i, v)$ depends on the complexity of the service chain SC_i which usually is given as *a priori*. Then, the end-to-end delay D_i of r_i via path P_i is the sum of the network delay of P_i and the processing delay of SC_i, i.e., $D_i = D_n(P_i) + D_p(i, v)$. It has to be guaranteed that $D_i \leq d_i$ for every admitted request r_i.

C. Problem Definition

Given an SDN $G = (V, E)$, a subset of switches V_{pm} ($\subseteq V$) with each attaching a PM of computing capacity C_v, the forwarding table capacity L_v for each switch $v \in V$, the bandwidth capacity B_e for each link $e \in E$, and a set of user requests $S(t)$ at time slot t, the *network throughput maximization problem* in the SDN G is to admit as many user requests as possible such that the throughput of numbers of admitted requests to the total number of requests in $S(t)$ is maximized, while the end-to-end delay d_i, the bandwidth demand b_i, and the computing demand $C(i, j)$ for service chain SC_i of each admitted request r_i and resource constraints of G are met.

D. NP-Hardness

We show that this problem is NP-hard by the following lemma.

Lemma 1. *The network throughput maximization problem in the software-defined network $G = (V, E)$ is NP-hard.*

The sketch of the proof: We show that the network throughput maximization problem in an SDN $G = (V, E)$ is NP-hard by a polynomial reduction from a version of *generalized assignment problem* (GAP) that is known to be NP-hard [5]. Given an instance of the GAP in the form of a set of bin $\mathcal{B}$, a set of items $\mathcal{I}$, bin capacities $cap\colon \mathcal{B} \mapsto \mathbb{R}^+$ and $size\colon \mathcal{B} \times \mathcal{I} \mapsto \mathbb{R}$, we first construct an SDN $G = (V, E)$, where $V = \mathcal{I} \cup \mathcal{B} \cup \{t\}$, $E = \{\langle n, m\rangle \mid n \in \mathcal{I}, m \in \mathcal{B}\} \cup \{\langle m, t\rangle \mid n \in \mathcal{B}\}$, and t is a virtual sink serving as the common destination for all requests. The forwarding table size of each node in V and the bandwidth resource capacity of each link in E are set to positive infinity. Moreover, $V_{pm} = \mathcal{B}$ and the computing capacity of each node m in V_{pm} is set to $cap(m)$, the capacity of bin m. We then generate a set of requests $S(t)$: For each item $n \in \mathcal{I}$, we add to $S(t)$ a request $r_i = \langle s_i, t_i, b_i, SC_i, d_i\rangle$, where s_i is set to the switch $n \in V$, t_i is set to the virtual sink t, $b_i = 0$, the computing resource demand $C(n, m)$ to process its service chain at $m \in V_{pm}$ is $size(n, m)$, and $d_i = \infty$. Therefore, routing the set of requests $S(t)$ into the network G is an instance of the network throughput maximization problem. We finish by noting that the network throughput maximization problem has a solution of admitting K requests if and only if the GAP with identical profits has a solution of profit K.

IV. Integer Linear Programming

In this section, we formulate the network throughput maximization problem as an Integer Linear Program (ILP). The detailed description is given in Figure 1, where x_i is a binary variable with value 1 if request r_i is admitted and 0 otherwise. z_i^v is a binary variable with value 1 if and only if the traffic of r_i is processed by the PM attached to switch $v \in V_{pm}$. For brevity, denote by $\delta^+(v)$ and $\delta^-(v)$ the sets of leaving and entering edges of a switch $v \in V$, respectively. In addition, to distinguish between traffic before and after being processed, we introduce two binary variables $w_i^{pre}(e)$ and $w_i^{post}(e)$ with value 1 if and only if link e carries the unprocessed and processed traffic, respectively.

Constraint (2) ensures that if and only if a request $r_i \in S(t)$ is admitted, it is processed in exactly one PM. Constraints (3) and (4) capture *traffic changing* at PM-attached switches that process traffic of user requests and *traffic conservation* at non-terminal switches. Specifically, if request r_i is processed at $v \in V_{pm}$, then (i) exactly one incoming edge of v carries the unprocessed traffic and none of the outgoing edges of v carries the unprocessed traffic; and (ii) exactly one of the outgoing edges of v carries the processed traffic, and none of the incoming edges of v carries the processed traffic. Otherwise, if the traffic of r_i is not processed by the PM attached to switch $v \in V_{pm}$ but goes through v, either (i) exactly one incoming edge and one outgoing edge of v carry the unprocessed traffic, or (ii) exactly one incoming edge and one outgoing edge of v carry the processed traffic. Constraints (6) and (7) handle the cases where the traffic of a request v_i is processed at the source switch s_i or the terminal switch t_i. Constraints (5) and (8) ensure that no unprocessed traffic enters any source switch s_i and no processed traffic leaves the terminal switch t_i. Constraint (9) enforces the end-to-end delay requirement. Constraint (10) enforces the bandwidth capacity constraint for each link $e \in E$, Constraint (11) imposes the forwarding table capacity constraint for each switch $v \in V$, and Constraint (12) models the computing capacity constraint of PMs. Constraints (14) to (16) restrict the range of decision variables. Constraint (17) indicates that if there is no PM at a switch $v \in V \setminus V_{pm}$, then it cannot process any request.

V. A Heuristic Algorithm

As the ILP solution is only applicable if the problem size is small, we here devise an efficient heuristic for the problem. We first propose a cost model to capture the dynamic resource usages of an SDN G, and then devise an algorithm for the problem by transforming it to a series of shortest paths finding in an auxiliary graph based on the proposed cost model.

$$maximize \quad \sum_{i=1}^{|S(t)|} x_i, \tag{1}$$

subject to

$$\sum\nolimits_{v\in V} z_i^v = x_i, \qquad i = 1,\ldots,|S(t)| \tag{2}$$

$$\sum\nolimits_{e\in\delta^-(v)} w_i^{pre}(e) - \sum\nolimits_{e\in\delta^+(v)} w_i^{pre}(e) = z_i^v, \qquad \forall v \in V \setminus \{s_i\}, \quad i = 1,\ldots,|S(t)| \tag{3}$$

$$\sum\nolimits_{e\in\delta^+(v)} w_i^{post}(e) - \sum\nolimits_{e\in\delta^-(v)} w_i^{post}(e) = z_i^v, \qquad \forall v \in V \setminus \{t_i\}, \quad i = 1,\ldots,|S(t)| \tag{4}$$

$$\sum\nolimits_{e\in\delta^-(s_i)} w_i^{pre}(e) = 0, \qquad i = 1,\ldots,|S(t)| \tag{5}$$

$$\sum\nolimits_{e\in\delta^+(s_i)} w_i^{pre}(e) = x_i - z_i^{t_i}, \qquad i = 1,\ldots,|S(t)| \tag{6}$$

$$\sum\nolimits_{e\in\delta^-(t_i)} w_i^{post}(e) = x_i - z_i^{t_i}, \qquad i = 1,\ldots,|S(t)| \tag{7}$$

$$\sum\nolimits_{e\in\delta^+(t_i)} w_i^{post}(e) = 0, \qquad i = 1,\ldots,|S(t)| \tag{8}$$

$$\sum\nolimits_{e\in E} \left(w_i^{pre}(e) + w_i^{post}(e)\right) + \sum\nolimits_{v\in V} z_i^v D_p(i,v) \le d_i, \qquad i = 1,\ldots,|S(t)| \tag{9}$$

$$\sum\nolimits_{i=1}^{|S(t)|} \left(b_i \cdot w_i^{pre}(e) + \lambda_i \cdot b_i \cdot w_i^{post}(e)\right) \le B_e, \qquad \forall e \in E \tag{10}$$

$$\sum\nolimits_{i=1}^{|S(t)|} \sum\nolimits_{e\in\delta^+(v)} (w_i^{pre}(e) + w_i^{post}(e)) \le L_v, \qquad \forall v \in V \tag{11}$$

$$\sum\nolimits_{i=1}^{|S(t)|} z_i^v \le C_v, \qquad \forall v \in V \tag{12}$$

$$w_i^{pre}(e) + w_i^{post}(e) \le 2, \qquad \forall e \in E, \quad i = 1,\ldots,|S(t)| \tag{13}$$

$$w_i^{pre}(e), w_i^{post}(e) \in \{0,1\}, \qquad \forall e \in E, \quad i = 1,\ldots,|S(t)| \tag{14}$$

$$x_i \in \{0,1\}, \qquad i = 1,\ldots,|S(t)| \tag{15}$$

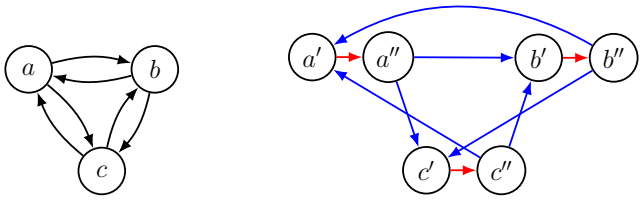
$$z_i^v \in \{0,1\}, \qquad \forall v \in V_{pm}, \quad i = 1,\ldots,|S(t)| \tag{16}$$

$$z_i^v = 0, \qquad \forall v \in V \setminus V_{pm}, \quad i = 1,\ldots,|S(t)|. \tag{17}$$

Fig. 1: An ILP formulation of the network throughput maximization problem

A. Cost Modeling and Auxiliary Graph Construction

Given an SDN $G = (V, E)$, the auxiliary graph $G' = (V', E'; \omega)$ is constructed, where $V' = \{v', v'' \mid v \in V\}$ and $E' = \{\langle v', v''\rangle \mid v \in V\} \cup \{\langle u'', v'\rangle \mid \langle u, v\rangle \in E\}$. Intuitively, an edge $\langle v', v''\rangle$ represents the switch v and an edge $\langle u'', v'\rangle$ represents link $\langle u, v\rangle$ in the network G. An example of the auxiliary graph construction is shown in Figure 2. Clearly, each



(a) $G = (V, E)$ (b) $G' = (V', E')$

Fig. 2: The auxiliary graph construction of G from G.

edge in G' represents either switch or link resources in G. A cost model of resource usages in network G is proposed as follows. For a given type of resource, the marginal cost of its usage dramatically inflates with the increase of its utilization ratio, since the larger the proportion of the resource is occupied, the higher the risk the resource capacity will be violated. We therefore use an exponential function to model the cost of resource usage. Denote by RL_v the residual capacity of the forwarding table at $v \in V$ and RB_e the residual bandwidth of link $e \in E$. Then, the weights of the corresponding edges in E' of switch node $v \in V$ and link $e \in E$ are:

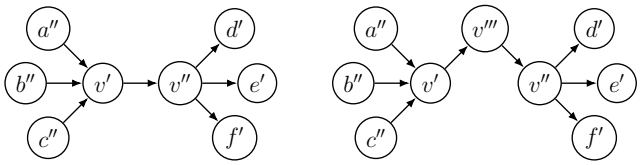
$$\omega(e) = \begin{cases} \alpha^{1-\frac{RL_v}{L_v}} & \text{if } e = \langle v', v''\rangle \in E', \\ \beta^{1-\frac{RB_{\langle v,u\rangle}}{B_{\langle v,u\rangle}}} & \text{if } e = \langle v'', u'\rangle \in E', \end{cases}$$

where α and β are constants with $\alpha, \beta > 1$. The larger the values of α and β, the more the resources with high utilizations will be discouraged from use, since its marginal cost will dramatically increase with further utilization.

B. Algorithm

The basic idea behind the proposed algorithm is to map different resource usages in the SDN G to the edge weights in the auxiliary graph G'. Then the problem in G is reduced to finding a series of shortest paths in G'. To admit a given

single user request $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle$ in $S(t)$, we first find a shortest path in $G' = (V', E')$ from s_i to t_i such that its corresponding routing path in G meets both bandwidth demand b_i and its end-to-end delay requirement d_i. Furthermore, a switch $v \in V_{pm}$ attached a PM in the path has sufficient computing resource to process the service chain SC_i of r_i. Specifically, we first remove the edges from G that do not have adequate resources. We then construct auxiliary graph $G'_i = (V'_i, E'_i)$ based on G. We now incorporate computing resources in PMs through augmenting G'_i for each PM-attached switch $v \in V_{pm}$, denoted by $G'_{i,v} = (V'_{i,v}, E'_{i,v})$. The only difference between $G'_{i,v}$ and G'_i is that the directed edge $\langle v', v'' \rangle$ ($\in E'_i$) is removed, and a new node v''' and edges $\langle v', v''' \rangle$ and $\langle v''', v'' \rangle$ are added to $V'_{i,v}$ and $E'_{i,v}$, respectively, as demonstrated in Figure 3 (b). Moreover, the weight of edge $\langle v''', v'' \rangle$ retains the weight of $\langle v', v'' \rangle$ in G'_i while the weight of $\langle v', v''' \rangle$ is $\gamma^{1-\frac{RC_v}{C_v}}$, where $\gamma > 1$ is a tuning parameter, RC_v is the residual computing capacity, and C_v is the capacity of v. Therefore, if $v \in V_{pm}$ is considered to process service chain SC_i of request r_i, routing the traffic of r_i is to find a path $P_i(v)$ in $G'_{i,v}$ that is the concatenation of a shortest path in $G'_{i,v}$ from s_i to v and a shortest path in $G'_{i,v}$ from v to t_i. Let $l(P_i(v))$ be the length of $P_i(v)$, i.e., $l(P_i(v)) = \sum_{e \in P_i(v)} \omega(e)$.



(a) Part of auxiliary graph $G' = (V', E')$

(b) Part of augmented auxiliary graph $G'_v = (V'_v, E'_v)$

Fig. 3: Augmenting auxiliary graph G' on the left to G'_v on the right for switch $v \in V_{pm}$

The problem of admitting a user request r_i in G is reduced to the problem of finding a shortest path $P_i(v)$ from all augmented auxiliary graphs $G'_{i,v}$ with the minimum length $\min\{l(P_i(v)) \mid \forall v \in V_{pm}\}$ and meets the end-to-end delay d_i. The detailed description of the algorithm is given in Procedure 1.

Having admitted a single request, we now consider the admission of a set of user requests $S(t)$ at time slot t. The idea is to admit the requests in $S(t)$ iteratively until no more requests can be admitted, where one request that incurs the minimum cost among the remaining requests will be admitted in each iteration. Specifically, in each iteration, Procedure 1 is employed to find a routing path for each remaining request r_i. Requests for which Procedure 1 fails to find routing paths will be rejected, and a request of which the found routing path has the minimum sum of edge weights will be admitted. This procedure repeats until every request in $S(t)$ is either rejected or admitted. The detailed description is given by Algorithm 1.

C. Algorithm Analysis

Theorem 1. *Given an SDN $G = (V, E)$ with a set V of switches and a set E of links, a subset $V_{pm} \subseteq V$ of switches each of which is attached with a PM, a set of user requests $S(t)$ at time slot t, there is an algorithm, Algorithm 1, for the network throughput maximization problem, which delivers a feasible solution in $O(|S(t)|^2|V|^4)$ time.*

Procedure 1 Admitting a single request $r_i \in S(t)$

Input: an SDN $G = (V, E)$ and a request $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle$
Output: a routing path $P_i = \langle s_i, \ldots, v \in V_{pm}, \ldots, t_i \rangle$ that satisfies b_i, SC_i, and d_i

1: Construct an auxiliary graph $G'_i = (V'_i, E'_i; \omega)$ for G after pruning resources without enough residual capacities;
2: $P_i^{sel} \leftarrow \infty$ /* a path in an augmented auxiliary graph with the minimum sum of edge weights */;
3: $l_{\min} \leftarrow \infty$ /* the minimum length of routing paths */;
4: **for each** PM-attached switch $v \in V_{pm}$ **do**
5: Augment G'_i to $G'_{i,v}$;
6: Let $P_i(v)$ be the concatenation of a shortest path in $G'_{i,v}$ from s'_i to v''' and a shortest path in $G'_{i,v}$ from v''' to t'_i;
7: **if** ($P_i(v)$ exists) **and** (its end-to-end delay is no more than d_i) **and** (its length is less than $l_{\min}$) **then**
8: Set P_i^{sel} to $P_i(v)$ and $l_{\min}$ to the length of $P_i(v)$;
9: **end if**
10: **end for**
11: If P_i^{sel} is found, find the corresponding routing path P_i in G;

Algorithm 1 A heuristic for routing a set of requests

Input: an SDN $G = (V, E)$ and a set of requests $S(t)$
Output: Routing decisions for requests in $S(t)$

1: $S' \leftarrow S(t)$ /* the set of requests to be admitted */;
2: **while** $S' \neq \emptyset$ **do**
3: **for each** request $r_i \in S'$ **do**
4: Use Procedure 1 to find a path P_i^{sel} for r_i. If such a path does not exist, reject r_i and remove it from S';
5: **end for**
6: Let $r_{\min}$ be a request of which the routing path found in Step 4 has the minimum weight among all requests;
7: Admit request $r_{\min}$ using the routing path found in Step 4, and update the resource availabilities of G;
8: $S' \leftarrow S' \setminus \{r_{\min}\}$;
9: **end while**

Proof: The solution delivered by Algorithm 1 is feasible because the auxiliary graphs are constructed from a subgraph of G that only includes resources with sufficient residual capacities. Consequently, the routing path in G converted from the path found in an augmented auxiliary graph G' is feasible.

We then analyze the time complexity of Algorithm 1. In Procedure 1, the construction and augmentation of the auxiliary graph take $O(|V| + |E|)$ time, while finding a shortest path in each of the $|V_{pm}|$ augmented auxiliary graphs takes $O(|V|^3)$ time. Procedure 1 thus takes $O(|V|^3 + |V| + |E|) = O(|V|^3)$ time. For each request $r_i \in S(t)$, Procedure 1 is invoked at most $|V_{pm}|$ times. The number of requests for which we need to find a shortest path is $O(|S(t)|^2)$. The time complexity of Algorithm 1 thus is $O(|S(t)|^2|V_{pm}||V|^3) = O(|S(t)|^2|V|^4)$. The theorem holds. ■

VI. A Faster Heuristic Algorithm

Although Algorithm 1 delivers a near optimal solution, its running time may still be high and may fail to respond to user

requests on time. We instead devise a faster heuristic to deal with dynamic user requests.

A. Overview of the Algorithm

A key ingredient of the proposed algorithm is that a candidate solution to admit a set $S(t)$ of requests is found, based on the residual capacities of SDN G in the beginning of time slot t, and no update of residual capacities is applied until all requests in $S(t)$ are considered. It then admits requests in $S(t)$ based on an auxiliary graph constructed from G and adjusts the admissions if the resource capacities of G are violated.

B. Algorithm

We first find a set of candidate routing paths $\mathcal{P}_i$ in G for each request $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle \in S(t)$, where a shortest path from s_i to t_i is considered as a candidate path of r_i as long as it has one PM-attached switch in V_{pm}, and satisfies b_i, SC_i, and d_i. Notice that we find candidate routing paths for requests in $S(t)$ on the augmented auxiliary graphs built based on the resource availability of network G as of the beginning of time slot t, through finding a shortest path from s'_i to v''' and a shortest path from v''' to t'_i in the auxiliary graph $G'_{i,v}$ for each request $r_i \in S(t)$ and $v \in V_{pm}$. Let $P_i(v_j) = \langle s_i, \ldots, v_j, \ldots, t_i \rangle$ be a found path in G_{i,v_j} for request r_i, whereas v_j ($\in V_{pm}$) is a switch that fulfills the service chain SC_i and $P_i(v_j)$ meets the resource and end-to-end delay constraints of r_i. Denote by $\mathcal{P}_i$ the set of candidate paths for request r_i, then, $\mathcal{P}_i = \{P_i(v_j) \mid v_j \in V_{pm}\}$.

Having the set of candidate paths $\mathcal{P}_i$ for every request r_i in $S(t)$, we then pick only one candidate path $P_i(v_j)$ from $\mathcal{P}_i$ for each request r_i such that the cost (length) sum of the selected paths is minimized, while ensuring that the computing capacity of each PM is not violated. We reduce this problem to the *Generalized Assignment Problem* (GAP), defined as follows. Given a set of items $\mathcal{I}$ and a set of bins $\mathcal{B}$, where each bin $m \in \mathcal{B}$ has a capacity $cap(b)$, each item $n \in \mathcal{I}$ has a size $size(n, m)$, and a profit $profit(n, m)$ if the item n is placed in bin m, the problem is to place a subset of items U ($\subseteq I$) in bins $\mathcal{B}$ such that the sum of the profits of items in U is maximized and the sum of sizes of items placed in every bin is no more than the capacity of the bin. Namely, we treat each PM-attached switch $v_j \in V_{pm}$ as a bin and each request r_i in $S(t)$ as an item, whereas the capacity of each bin v_j is its residual computing capacity, i.e., $cap(v_j) = LC_{v_j}$, the size of an item r_i in a bin v_j is the computing demand of the service chain SC_i in the PM attached to v_j, i.e., $size(r_i, v_j) = C(i, j)$, and the profit of placing an item r_i in a bin v_j is the reciprocal of the length of the candidate path that fulfills r_i on v_j, i.e., $profit(i, j) = \frac{1}{l(P_i(v_j))}$.

Having reduced the network throughput maximization problem to the GAP, we now solve the GAP and each solution to the GAP yields a solution to the original problem. Specifically, we use the algorithm proposed by Cohen *et al.* [5] that guarantees a $(2 - \epsilon)$-approximation ratio, where ϵ is a constant with $0 < \epsilon \leq 1$, to solve the GAP. Denote by U a solution found by this algorithm as a placement of a subset of items in bins. U yields a potential admission of requests in $S(t)$: for every request r_i treated as an item, if it is placed in a bin representing $v_j \in V_{pm}$, then it is admitted with the routing path $P_i(v_j)$; otherwise, r_i is rejected.

Due to the construction of the GAP, admitting requests in $S(t)$ based on the solution U to the GAP ensures that the sum of computing demands of requests of which the service chains are fulfilled in a same PM will not exceed the computing capacity of the PM. However, the bandwidth and forwarding table capacities may be violated, as routing paths may have overlapping resources. We thus perform adjustments to eliminate such potential resource violations by selectively rejecting some requests. Let $P_i^{sel} = P_i(v_j) = \langle s_i, \ldots, v_j, \ldots, t_i \rangle$ be the path to route the traffic of request r_i according to U, where $v_j \in V_{pm}$. We build a bipartite graph $G_b = (U_b, V_b, E_b)$ with selected routing paths for all potentially admitted requests as node set U_b, and auxiliary edges whose resource capacities will be violated if admissions are indeed performed, as another node set V_b. There is an edge between a node $P_i^{sel} \in U_b$ and a node $e \in V_b$ if e is in path P_i^{sel}. The weight of edge (P_i^{sel}, e) is the ratio of the demand of r_i on that resource to the sum of those of all requests on that resource, which represents the contribution of r_i to the resource capacity violation of e. To eliminate the violations, we iteratively remove one node P_i^{sel} with the largest accumulative weight of incident edges in G_b from U, and update G_b by removing nodes in V_b that their resource overloadings are avoided due to the removal of node P_i^{sel}. This procedure repeats until no edge is left in E_b. The detailed description is given in Algorithm 2.

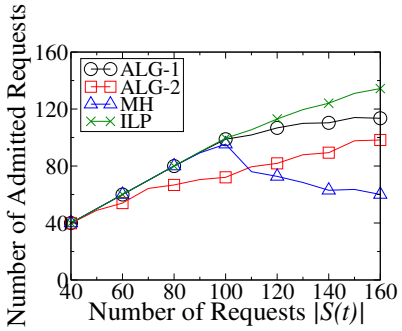
Algorithm 2 A faster heuristic for routing a set of requests $S(t)$ into a software-defined network G

Input: an SDN $G = (V, E)$ and a set of user requests $S(t)$
Output: Routing decisions for each request $r_i \in S(t)$

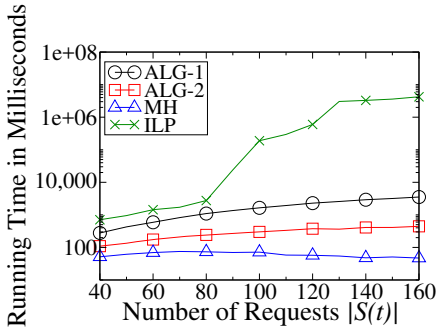
```
Build an auxiliary graph G' = (V', E') for G;
Initialize P, the set of candidate routing paths in G for all requests in S(t), to ∅;
for each user request r_i ∈ S(t) do
    P_i ← ∅; /* the set of candidate paths for request r_i */
    for each PM-attached switch v_j ∈ V_pm do
        Use Procedure 1 to find a path P_i(v_j) that contains v_j
        If P_i(v_j) exists, add it to P;
    end for
    If P_i is empty, reject the request. Otherwise, add P_i to P;
end for
Construct an instance of the GAP by representing each request as an item and each node in V_pm as a bin;
Solve the GAP instance using the algorithm in [5]
Construct a bipartite graph G_b = (U_b, V_b, E_b) that reflects potential capacity violations;
while there are edges in E_b do
    Remove a node in U_b with the maximum weighted sum of incident edges and its incident edges from G_b;
    Update G_b;
end while
```

C. Algorithm Analysis

Theorem 2. *Given an SDN $G = (V, E)$ with a set V of switches and a set E of links, a subset $V_{pm} \subseteq V$ of switches*



(a) The numbers of requests admitted by different algorithms



(b) Running time of different algorithms in milliseconds on a logarithmic scale

Fig. 4: Performance of ALG−1, ALG−2, MH, and ILP in the GÉANT topology within a single time slot.

each of which is attached with a PM, a set of user requests $S(t)$, there is an algorithm for the network throughput maximization problem, Algorithm 2, which takes $O(|S(t)||V|^3 + |V| \cdot \frac{|S(t)|^3}{\epsilon})$ time, where ϵ is a given constant with $0 < \epsilon \leq 1$.

Proof: Algorithm 2 consists of three phases: (i) find a set of candidate routing paths for each request; (ii) select only one routing path for each request to meet computing capacities of nodes in V_{pm}; and (iii) eliminate the requests that violate bandwidth or forwarding table capacities. The feasibility of solutions delivered by Algorithm 2 follows from the Phase (ii). Phase (i) takes $O(|S(t)||V|^3)$ time because $O(|V_{pm}|) = O(|V|)$ shortest paths are found for each request $r_i \in S(t)$ in augmented auxiliary graphs and each shortest path takes $O(|V|^2)$ time to find. The running time of Phase (ii) is dominated by the time required to solve the GAP, which is $O(|V| \cdot \frac{|S(t)|^3}{\epsilon})$ [5]. Phase (iii) takes $O(|S(t)|(|V|+|E|))$ time, there are $O(|S(t)|(|V|+|E|))$ edges in the bipartite graph, and in the worst case, each request violates the resource capacities of all switches and links. The theorem thus holds. ■

VII. Performance Evaluation

In this section, we evaluate the performance of proposed algorithms using experimental simulations based on several real network topologies.

A. Experimental Environment

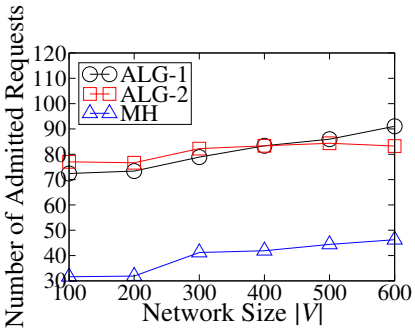
We adopt commonly used, real network topologies including GÉANT [16] and several ISP networks from [21] in the simulations, where GÉANT [16] is a European network consisting of 40 nodes and 122 links. The size of the forwarding table of each switch is from 1,000 to 8,000 [13]. The bandwidth of each Internet link varies from 1,000 Mbps to 10,000 Mbps [12]. There are nine PMs for the GÉANT topology as set in [9] and the number of PMs in ISP networks are provided by [18]. The computing capacity of each PM is from 4,000 to 8,000 MHz [10]. The delay of an Internet link is between 2 milliseconds (ms) and 5 ms [12], [13]. We consider five types of middleboxes: Firewall, Proxy, NAT, IDS, and Load Balancing, and their computing demands are adopted from [9], [16]. The running time is obtained based on a machine with a 3.40GHz Intel i7 Quad-core CPU and 16 GiB RAM. The default accuracy parameter ϵ in solving GAP is set to 0.1. Unless otherwise specified, these parameters will be adopted in the default setting. Each request $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle \in S(t)$ is generated as follows, given a network $G = (V, E)$, two nodes from V are randomly drawn as the source switch s_i and the destination switch t_i of request r_i. The bandwidth demand b_i is randomly drawn from 10 to 120 Mbps [1] and the delay varies from 40 ms to 400 ms [17].

We evaluate Algorithms 1 and 2 against a baseline heuristic which is described as follows. Sort all requests in $S(t)$ in non-decreasing order by their computing resource demands, and then, for each request $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle$ in $S(t)$, find a shortest path from s_i to a PM-attached switch v ($\in V_{pm}$) with the minimum number of hops from s_i and a shortest path from v to d_i. We refer to the minimum-hop-based baseline, Algorithm 1, and Algorithm 2 as MH, ALG−1, and ALG−2, respectively. Each value in figures is the mean of the results of 30 trials.

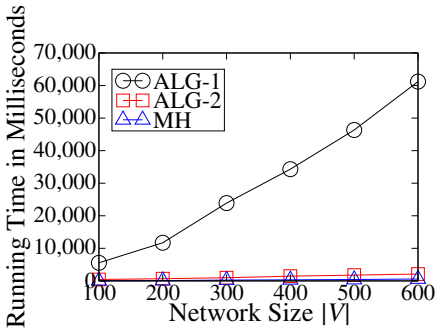
B. Algorithm Performance within a Single Time Slot

We first investigate the performance of the proposed algorithms in the GÉANT topology within a single time slot.

Fig. 4 (a) shows the numbers of requests admitted by different algorithms, when the number of requests in a time slot is a constant in the range from 40 to 160. It can be seen that both ALG−1 and MH can admit as many requests as ILP does if there are less than 100 requests. Otherwise, only ALG−1 can achieve a comparable throughput as ILP. This means that the network throughput of ALG−2 is inferior to ALG−1 and the difference between them enlarges from nearly zero at $|S(t)| = 40$ to 21 at $|S(t)| = 160$. The reason is that ALG−2 will reject more requests with the increase in the number of requests, as the likelihood of routing paths that ALG−2 finds for different requests being overlapping and resource being violated soars. Meanwhile, we notice that MH outperforms ALG−2 if the number of requests is small. Otherwise, ALG−2 outperforms MH. Specifically, when there are 160 requests, the number of requests admitted by MH is only 60% of the one by ALG−2 and runs much faster than the latter. The reason behind

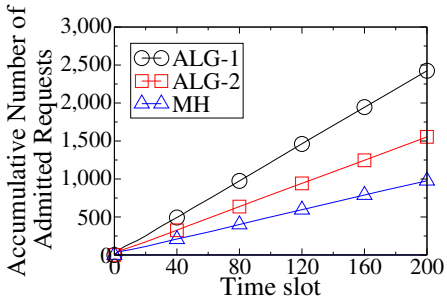


(a) The numbers of requests admitted by different algorithms

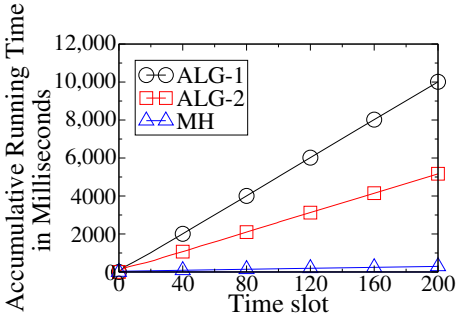


(b) Running time of different algorithms in milliseconds

Fig. 5: Performance of ALG−1, ALG−2, and MH in the GÉANT network when the number of switches varies from 100 to 600 while the number of requests is fixed at 160 per time slot.



(a) Total number of requests admitted by different algorithms



(b) Accumulative running time of algorithms

Fig. 6: Performance of ALG−1, ALG−2, and MH on a GÉANT network for a time horizon with 200 time slots where the number of requests in each time slot follows a Poisson distribution with a mean of 30.

this is that MH does not guarantee that the overall path from its source s_i of a request r_i to its destination t_i has the minimum weight, since it finds shortest paths from s_i to a PM-attached switch and from the PM-attached switch to t_i in two separate stages. Fig. 4 (b) illustrates the amount of time spent by these algorithms, from which we can see that the running time of the ILP is orders of magnitude slower than those of the other algorithms, while ALG−2 is significantly faster than ALG−1 and MH is the fastest.

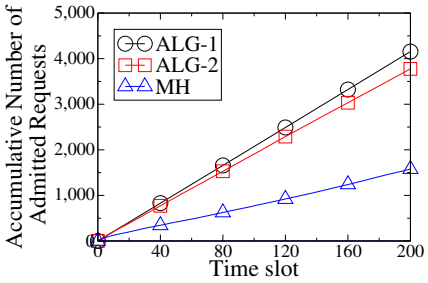
We then evaluate the performance of different algorithms by varying the network size. As topologies such as [12], [21] have limited sizes, we adopt the widely used Barabási-Albert model [2] to generate networks of different sizes. Namely, we vary the number of switches in an SDN from 100 to 600, while fixing the number of requests at 160. The results are depicted in Fig. 5. We can see from Fig. 5 (a) that ALG−1 and ALG−2 achieve the similar throughput, while MH admits no more than a half as requests as two heuristics. Fig. 5 (b) also reveals that ALG−2 runs much faster than ALG−1.

C. Algorithm Performance within a Finite Time Horizon

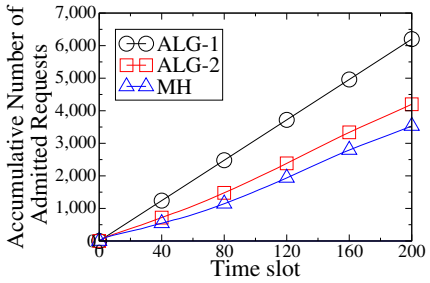
We now consider a time horizon consisting of 200 time slots, under which we evaluate different algorithms, assuming that the number of requests at each time slot follows a Poisson distribution with a mean of 30, and each admitted request spans 1 to 10 time slots randomly.

The results are summarized in Fig. 6. From Fig. 6 (a), we can see that MH has the lowest network throughput among the three algorithms. On the other hand, ALG−1 and ALG−2 utilize resources more efficiently, and hence they can admit much more requests than that of MH by 150% and 50%, respectively. From Fig. 6 (b), it can be seen that the running time of MH is negligible compared with those of ALG−1 and ALG−2. It must be noticed that this running time comes at the cost of admitting much fewer requests.

We finally evaluate ALG−1, ALG−2, and MH on three Autonomous Systems (AS) from [21]: AS-4755 with 121 switches and 296 links, and AS-1755 with 172 switches and 762 links. We consider a time horizon consisting of 200 time slots and the number of requests at each time slot follows a Poisson distribution with a mean of 30. The results are illustrated in Fig. 7, from which it can be seen that in all topologies, ALG−1 is the best, while MH is the worst. The performance gap between ALG−1 and ALG−2 is small compared to that in the GÉANT topology, since the size of these three topologies is larger than that of the GÉANT topology, and routing paths delivered by



(a) The accumulative numbers of requests admitted by different algorithms for AS-4755



(b) The accumulative numbers of requests admitted by different algorithms for AS-1755

Fig. 7: The accumulative numbers of admitted requests of ALG−1, ALG−2, and MH for a time horizon consisting of 200 time slots in ISP networks.

ALG−2 for different requests in a time slot are less likely to overlap. In AS-4755, the difference on requests admitted by ALG−1 and ALG−2 is less than 500, while MH admits no more than 40% requests of the other two algorithms. On the other hand, the performance of MH in AS-1755 improves due to the larger network capacity.

VIII. Conclusions

In this paper, we studied the problem of realizing user requests with each specifying a sequence of middleboxes in an SDN, with the objective to maximize the network throughput, subject to the constraints of forwarding table capacity, network bandwidth capacity, computing resource capacity, and user QoS requirements. We first formulated an ILP solution to the problem when the problem size is small. We then devised two heuristic algorithms that strive for fine tradeoffs between the accuracy of the solutions and the running times of the proposed algorithms. We finally evaluated the performance of the proposed algorithms by simulations using real network topologies and synthetic traffic traces. Experimental results demonstrated that the proposed algorithms are very promising.

References

[1] Amazon Web Services, Inc. Amazon ec2 instance configuration. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-ec2-config.html.

[2] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, Vol. 286, pp. 509–512, 1999.

[3] M. Charikar *et al.* Multi-commodity flow with in-network processing. http://www.cs.princeton.edu/~jrex/papers/mopt14.pdf, 2014.

[4] C.-H. Chi, J. Deng, and Y.-H. Lim. Compression proxy server: design and implementation. *Proc. of USITS*, 1999.

[5] R. Cohen, L. Katzir, and D. Raz. An efficient approximation for the generalized assignment problem. *Information Processing Letters*, Vol. 100, pp. 162–166, Elusive, 2006.

[6] A. Gember-Jacobson *et al.* Stratos: a network-aware orchestration layer for middleboxes in the cloud. *arXiv:1305.0209*, 2013.

[7] A. Gember-Jacobson *et al.* OpenNF: enabling innovation in network function control. *Proc. of SIGCOMM*, ACM, 2014.

[8] A. Gupta *et al.* SDX: a software defined internet exchange. *Proc. of SIGCOMM*, ACM, 2014.

[9] A. Gushchin, A. Walid, and A. Tang. Scalable routing in sdn-enabled networks with consolidated middleboxes. *Proc. HotMiddlebox'15*, ACM.

[10] Hewlett-Packard Development Company, L.P. Servers for enterprise – bladeSystem, rack & tower and hyperscale. http://www8.hp.com/us/en/products/servers/, 2015.

[11] M. Honda *et al.* Is it still possible to extend TCP? *Proc. IMC'11*, ACM.

[12] S. Knight *et al.* The internet topology zoo. *J. Selected Areas in Communications*, Volume 29, pp. 1765–1775, IEEE, 2011.

[13] D. Kreutz *et al.* Software-Defined Networking: a comprehensive survey. *Proceedings of IEEE*, Volume 103, pp. 14–76, IEEE, 2015.

[14] T.-W. Kuo *et al.* Deploying chains of virtual network functions: On the relation between link and server usage. *Proc. of INFOCOM'16*, IEEE.

[15] Y. Li, L. T. X. Phan, and B. T. Loo. Network function virtualization with soft real-time guarantees *Proc of INFOCOM'16*, IEEE.

[16] J. Martins *et al.* ClickOS and the art of network function virtualization. *Proc. of NSDI*, 2014.

[17] Microsoft. Plan network requirements for Skype for business. https://technet.microsoft.com/en-us/library/gg425841.aspx, 2015.

[18] Z. A. Qazi *et al.* SIMPLE-fying middlebox policy enforcement using SDN. *Proc. of SIGCOMM*, ACM, 2013.

[19] V. Sekar *et al.* Design and implementation of a consolidated middlebox architecture. *Proc. of NSDI*, USENIX, 2012.

[20] J. Sherry *et al.* Making middleboxes someone else's problem: network processing as a cloud service. *Proc. of SIGCOMM*, ACM, 2012.

[21] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with rocketfuel. *Proc. of SIGCOMM*, ACM, 2002.