Efficient Algorithms for Throughput Maximization in Software-Defined Networks With Consolidated Middleboxes

Meitian Huang, Weifa Liang, *Senior Member, IEEE*, Zichuan Xu, *Member, IEEE*, and Song Guo, *Senior Member, IEEE*

***Abstract*—Today's computer networks rely on a wide spectrum of specialized middleboxes to improve network security and performance. A promising emerging technique to implementing traditional middleboxes is the consolidated middlebox technique, which implements the middleboxes as software in virtual machines in software-defined networks (SDNs), offering economical, and simplified management for middleboxes. This however poses a great challenge, that is, how to find a cost-optimal routing path for each user request such that the data traffic of the request will pass through the middleboxes in their orders in the service chain of the request, with the objective to maximize the network throughput, subject to various resource capacity constraints in SDNs. In this paper, we study the network throughput maximization problem in an SDN under two different scenarios: one is the snapshot scenario where a set of requests at one time slot is given, we aim to admit as many requests in the set as possible to maximize the network throughput; another is the online scenario in which requests arrive one by one without the knowledge of future arrivals. Given a finite time horizon consisting of T equal time slots, the system must respond to the arrived requests in the beginning of each time slot, by either admitting or rejecting the requests, depending on the resource availabilities in the network. For the snapshot scenario, we first formulate an integer linear program (ILP) solution, we then devise two heuristics that strive for fine tradeoffs between the quality of a solution and the running time of obtaining the solution. For the online scenario, we show how to extend the proposed algorithms for the snapshot scenario to solve the online scenario. We finally evaluate the performance of the proposed algorithms through experimental simulations, based on both real and synthetic network topologies. Experimental results demonstrate that the proposed algorithms admit more requests than the baseline algorithm and the quality of the solutions delivered by heuristics is comparable to the exact solution by the ILP in most cases.**

***Index Terms*—Software-defined networking, network function virtualization, consolidated middleboxes, throughput maximization, routing algorithms, online algorithms, network resource allocations.**

Manuscript received March 13, 2017; revised July 5, 2017; accepted July 6, 2017. Date of publication July 11, 2017; date of current version September 7, 2017. The associate editor coordinating the review of this paper and approving it for publication was V. Fodor. *(Corresponding author: Weifa Liang.)*

M. Huang and W. Liang are with the Research School of Computer Science, Australian National University, Canberra, ACT 2601, Australia (e-mail: u4700480@anu.edu.au; wliang@cs.anu.edu.au).

Z. Xu is with the School of Software, Dalian University of Technology, Dalian 116024, China (e-mail: zichuanxu.mail@gmail.com).

S. Guo is with the Department of Computing, Hong Kong Polytechnic University, Hong Kong (e-mail: song.guo@polyu.edu.hk).

Digital Object Identifier 10.1109/TNSM.2017.2725240

I. Introduction

COMPUTER networks nowadays rely on various middleboxes, including firewall, Intrusion Detection Systems (IDSs), WAN optimizers, and Deep Packet Inspections (DPIs), to enhance the performance and security of different network services [10], [14], [28]. Unfortunately, the management and deployment of these hardware middleboxes are complex and costly [28]. For example, statistics indicated that large networks (10k-100k nodes) spent over a million dollars on deploying and maintaining hardware middleboxes while medium and small networks (1k-10k nodes) spent between $5,000 and $50,000 in the last five years [28]. With the advancement of the Network Function Virtualization (NFV), middleboxes can be implemented in Virtual Machines (VMs) that run in Physical Machines (PMs) [9], [26], [28]. The NFVs can be relocated and instantiated at servers located at different locations in a network without needs of purchasing and installing expensive middleboxes. By decoupling network functions from the hardware platform on which network functions are executed, NFV has the great potential to lead to significant reductions in operating expenses (OPEX) and capital expenses (CAPEX) of network service providers and facilitate the deployment of new services with increased agility and faster time-to-value [25]. We refer to the software implementation of middleboxes as the *consolidated middleboxes*. Along with the technique of Software-Defined Networking (SDN), consolidated middleboxes offer a promising alternative way to provide cheap and simplified management of middleboxes [12], [27].

In this paper we deal with realizing user requests with each specifying a sequence of middleboxes in SDNs with the aim to maximize the network throughput. This problem poses great challenges. One challenge is that different types of resources in SDNs have different 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. TCAM however is expensive and energy hungry, its capacity thus is restricted to a few thousand table entries [18]. Meanwhile, the computing resource of the PM attached to an SDN-enabled switch is limited too. Another challenge is that all resources in an SDN are dynamically allocated, causing significant fluctuations in their consumptions and availabilities. The 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. In this paper we will address the aforementioned challenges.

In spite of several studies of consolidated middleboxes [5], [12], [26], none of the studies has taken the forwarding table size into consideration. Almost all existing solutions adopt a strategy that decomposes the routing path finding and the service chain execution into two separate subtasks [12], the solutions thus are suboptimal. To the best of our knowledge, we are the first to formulate a novel routing optimization problem with consolidated middleboxes in SDNs by jointly taking into account both routing path finding and consolidated middlebox placement while meeting different user QoSs, by providing efficient heuristic solutions.

The main contributions of this paper are as follows. We consider the network throughput maximization problem of realizing user requests with service chains in SDNs, subject to various network resource capacity constraints. We first formulate an Integer Linear Program (ILP) solution to the problem when the problem size is small. We then devise a heuristic by providing a novel cost model to capture resource consumptions. We also propose a faster heuristic to quickly respond to user requests, by exploring non-trivial tradeoffs between the accuracy (quality) of a solution and the running time of obtaining the solution. Furthermore, we consider dynamic admissions of user requests where user requests arrive one by one without the knowledge of future arrivals, by showing how to extend the proposed algorithms for dynamic admissions of requests. We finally evaluate the performance of the proposed algorithms through simulations, based on real and synthetic network topologies. Experimental results demonstrate that the proposed algorithms are very promising compared to a baseline algorithm and the ILP, which delivers optimal solutions.

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 two heuristic algorithms. Section VII will devise an online algorithm for dynamic request admissions. Section VIII will evaluate the performance of the proposed algorithms through simulations, and Section IX will conclude the paper.

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 and operational costs [27], [28]. To tackle these issues, recent efforts on new frameworks and architectures of consolidated middleboxes [2], [9], [11], [23], [27], have been demonstrated as promising alternatives to traditional hardware middleboxes. For example, Sekar *et al.* [27] devised an architecture CoMb that focused on consolidating software-based implementations of middlebox functions on a shared hardware platform. Qazi *et al.* [26] developed SIMPLE that enforces high-level routing policies for middlebox-specific traffic. Fayazbakhsh *et al.* [8] proposed FlowTags, because traditional flow rules do not suffice in the presence of dynamic modifications performed by middleboxes. Martins *et al.* [23] introduced a virtualization platform to improve network performance by revising existing virtualization technologies to support the deployment of modular, virtual middleboxes on lightweight VMs.

One fundamental problem under such architecture is network throughput maximization of realizing user routing requests with specified service chains while meeting various resource constraints and user QoS requirements. A few recent studies investigated this issue [5], [12], which however neither considered resource constraints such as the forwarding table size constraint on switches, nor took global optimization, thereby the solutions delivered are suboptimal specifically, Charikar *et al.* [5] assumed that every switch in a network can perform middlebox functions without considering forwarding table sizes. Gushchin *et al.* [12] assumed that the routing traffic of a request can be split into multiple paths, and proposed a two-stage local optimization (before and after the virtual middleboxes). Zhang *et al.* [33] presented a routing scheme that reduces TCAM space usage without causing network congestion. However, they did not consider user requests with service chain requirements. Kuo *et al.* [19] studied a problem of VM placement and path selection, striving 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.* [20] presented the design and implementation of a system that dynamically provisions resources to provide timing guarantees with the objective of maximizing the number of requests admitted to the cloud, while meeting the deadline requirements of admitted requests. In another related paper [15], Huang *et al.* considered a joint optimization problem of middlebox selection and routing with the objective to maximize the throughput or a specified set of sessions in an SDN, and proposed a polynomial algorithm based on the Markov approximation technique. Cao *et al.* [4] studied the problem of policy-aware traffic engineering in SDNs, by assuming that the traffic has to pass a given sequence of network functions. Lukovszki *et al.* [21] recently considered middlebox placements in a n-node network so that each source-destination pair in a given set has a path of length at most L with one middlebox in it, and each middlebox can be used by at most k pairs. They devised an approximation algorithm with an approximation ratio of $O(\log \min\{n, k\})$ for the problem, under the assumption that only one VM (or a network function) is associated with each request, and each server can accommodate no more than k VMs. Clearly, this assumption is over-simplified as the length of a service chain of each request may be far greater than one. Lukovszki and Schmid [22] achieved several important theoretical results under ideal assumptions that each server can accommodate only one VM and different VMs for different network functions consume the

same amount of computing resource. Xu *et al.* [32] devised the very first approximation algorithm for the NFV-enabled multicasting problem and online algorithm with a competitive ratio for dynamic admissions of NFV-enabled multicast requests without the knowledge of future request arrivals.

Following the same assumption in [5] and [12], in this paper we assume that middleboxes are implemented as software applications running as VMs in servers or data centers. Meanwhile, there are many possibilities for resource sharing, one of which is to use a dedicated VM for each NFV. However, considering a chain of network functions is often made up of several functions [28], this approach will clearly not be feasible as physical resources will easily be depleted, and will be wasteful of resources since most functions are light-weight and can therefore be processed by a single VM, e.g., by containers within the VM [9]. Therefore, we further adopt the idea of consolidated middleboxes [12], where every flow obtains all its required functional treatment at a single PM, because the consolidated middlebox model simplifies traffic routing, helps reduce the number of routing rules in the switches, and removes the topology dependence between different middleboxes. On the other hand, in contrast to [5], we do not allow routing traffic via multiple paths from its source to its destination, because most network functionalities will be applied to the entire packet flow, e.g., encryption and decryption should only be applied to an entire message.

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 a 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 is connected to physical machines (PMs) to implement middleboxes as virtual machines. As such a switch and its attached PM usually are connected by a high-speed optical link, the latency between them is negligible. In the rest of this paper, the switches and their attached PMs will be used interchangeably. Denote by V_{pm} ($\subseteq V$) the set of switches that have attached PMs. Without loss of generality, we assume that each PM attached to a switch $v \in V_{pm}$ has limited computing resource capacity, denoted by C_v. 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 logically 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 the 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 request r_i therefore involves routing the traffic from the source switch s_i to the destination switch t_i via a routing path $P_i = \langle s_i, \ldots, t_i\rangle$ subject to the specified constraints.

Following the same assumption as in [12], [23], [26], and [27], 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, the traffic 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 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 to the traffic, while the volume of traffic decreases if compression is applied to the traffic. We here define $\lambda_i \in \mathbb{R}^+$ as the ratio between the volumes of the traffic of request r_i before and after processing at a PM. Since request r_i requires an amount b_i of bandwidth to route its traffic before processing, it 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 traces of similar requests [6]. In addition, each request r_i has a tolerant end-to-end delay requirement d_i. Suppose that request 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 $d(P_i)$ and $d(i, v)$ 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 $d(P_i)$ is proportional to the number of switches on P_i, and the average processing delay $d(i, v)$ 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., $d(P_i) + d(i, v)$. It has to be guaranteed that $d(P_i) + d(i, v) \leq d_i$ 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 with 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 G is to admit as many user requests in $S(t)$ as possible such that the number of requests admitted is maximized while the end-to-end delay d_i, bandwidth demand b_i, and computing demand $C(i, j)$ of the service chain SC_i of each

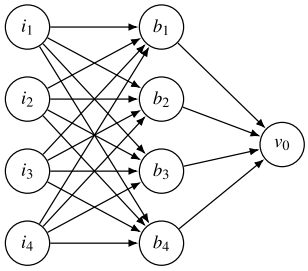


Fig. 1. An example of the SDN G constructed from an instance of the GAP with four items and four bins.

admitted request $r_i \in S(t)$ is met, subject to resource capacity constraints in G.

Given an SDN $G = (V, E)$, a subset of switches V_{pm} ($\subseteq V$) with each attaching a PM with 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 given time horizon T that consists of T equal time slots, assume that the set of requests arrived at time slot t is $S(t)$ and the duration of each request $r_i = \langle s_i, t_i, b_i, SC_i, d_i, \tau_i \rangle \in S(t)$ in the system is τ_i time slots with $1 \leq \tau_i \leq \tau_{\max}$, the *online network throughput maximization problem* in G is to admit as many user requests as possible during time horizon T such that the number of requests admitted is maximized while the end-to-end delay d_i, bandwidth demand b_i, and computing demand $C(i, j)$ of the service chain SC_i of each admitted request $r_i \in S(t)$ is met, subject to resource capacity constraints in G.

D. NP-Hardness

We show that the network throughput maximization problem is NP-hard by the following lemma.

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

Proof: We show that the network throughput maximization problem in $G = (V, E)$ is NP-hard, by a polynomial reduction from the *generalized assignment problem* (GAP) which is a well-known NP-hard problem [7]. Given an instance of the GAP in the form of a set of bins $\mathcal{B} = \{b_1, \ldots, b_n\}$, a set of items $I = \{i_1, \ldots, i_m\}$, bin capacities cap: $\mathcal{B} \mapsto \mathbb{R}^+$ and $size$: $\mathcal{B} \times I \mapsto \mathbb{R}^+$. For each item i_j with $1 \leq j \leq m$ and bin b_k with $1 \leq k \leq n$, we are given a size $size(j, k)$ and a profit $profit(j, k)$. The problem is to pack a subset $U \subseteq I$ of items to the bins in $\mathcal{B}$ such that the total profit by these items is maximized. The GAP problem is a well-studied problem.

We first construct an SDN $G = (V, E)$, through adding a stand-alone switch i for each item i in I, a PM-attached switch b for each bin b in $\mathcal{B}$, a virtual sink v_0 that is serving as the common destination for all requests, a link from each stand-alone switch to each PM-attached switch, and a link from each PM-attached switch to the virtual sink v_0. That is, $V = I \cup \mathcal{B} \cup \{v_0\}$ and $E = \{\langle i, b \rangle \mid i \in I, b \in \mathcal{B}\} \cup \{\langle b, v_0 \rangle \mid b \in \mathcal{B}\}$. Fig. 1 shows an example of the constructed SDN $G = (V, E)$.

The forwarding table size at each node in V and the bandwidth resource capacity of each link in E are set to 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: For each item $n \in I$, we add to S 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 into 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 Program

In this section, we formulate the network throughput maximization problem as an Integer Linear Program (ILP), where x_i is a decision variable with value 1 if request r_i is admitted and 0 otherwise. z_i^v is a decision 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 at a PM, we introduce two decision 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. The detailed description is given in Fig. 2,

Constraint (2) ensures that if and only if a request $r_i \in S(t)$ is admitted, it will be processed in exactly one PM. The volume of the traffic may change after the processing at v, while the volume is conserved at other non-terminal switches except the switch $v \in V_{pm}$ where it is processed.

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 (5) and (8) ensure that no unprocessed traffic enters any source switch s_i and no processed traffic leaves the terminal switch t_i.

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.

Constraint (9) enforces that the end-to-end delay requirement, which is the sum of the network delay $D_n(P_i)$ and the processing delay $D_p(i, v)$, of every admitted request is met, where the network delay $D_n(P_i)$ is calculated by $\sum_{e \in E}(w_i^{pre}(e) + w_i^{post}(e))$ and the processing delay $D_p(i, v)$ is $\sum_{v \in V} z_i^v \cdot D_p(i, v)$. Since z_i^v is 1 only for node v that implements the consolidated middleboxes for request r_i, only the processing delay at the node v is incurred.

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

subject to

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

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

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

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

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

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

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

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

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

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

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

$$w_i^{pre}(e), w_i^{post}(e) \in \{0, 1\}, \qquad \forall e \in E, \quad i = 1, \ldots, |S(t)| \tag{13}$$

$$x_i \in \{0, 1\}, \qquad i = 1, \ldots, |S(t)| \tag{14}$$

$$z_i^v \in \{0, 1\}, \qquad \forall v \in V_{pm}, \quad i = 1, \ldots, |S(t)| \tag{15}$$

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

Fig. 2. An ILP formulation of the network throughput maximization problem.

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 attached to each switch $v \in V_{pm}$.

Constraints (13), (14), and (15) restrict the range of decision variables to 0 and 1 inclusively. Constraint (16) indicates that if there is no PM at a switch $v \in V \setminus V_{pm}$, then it cannot process any request.

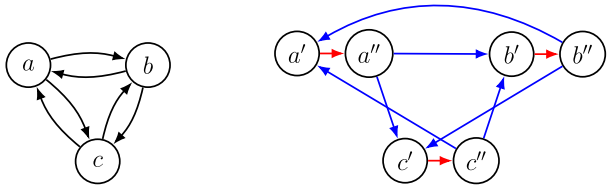
Since the ILP solution is time-consuming, it is only applicable when the problem size is small. The rest of this paper will develop efficient, scalable solutions to the problem.

V. A Heuristic Algorithm

In this section, we focus on devising an efficient heuristic for the problem. We first propose a cost model to capture the dynamic resource usages in G, and then devise the algorithm through a reduction that reduces the problem into shortest path findings in a series of auxiliary graphs derived from G.

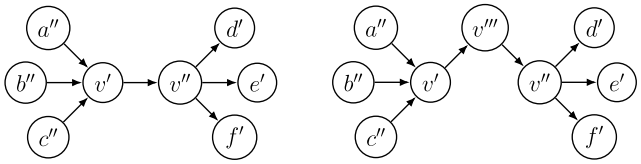
A. A Novel Cost Model of Resource Usages and the Construction of An Auxiliary Graph

Given an SDN G, it contains different types of resources such as computing resources at servers, TCAM sizes at switches, and bandwidth resources at links. Designing an efficient algorithm for the network throughput maximization problem needs to utilize these resources judiciously, through the guidance of an efficient cost metric that can accurately capture the usages and utilizations of different resources. In the following, we first propose a cost model of resource usages. We then reduce the problem of concern in G into another problem of finding shortest paths in a series of auxiliary graphs G_i' that are derived by implementing the service chain at different servers in G.



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

Fig. 3. The auxiliary graph construction of G'_i from G for the ith request: (a) the original SDN $G = (V, E)$; and (b) the corresponding auxiliary graph $G'_i = (V'_i, E'_i)$ of G.



(a) Part of auxiliary graph $G'_i = (V'_i, E'_i)$ (b) Part of augmented auxiliary graph $G'_{i,v} = (V'_{i,v}, E'_{i,v})$

Fig. 4. Augmenting auxiliary graph G'_i on the left to $G'_{i,v}$ on the right for switch $v \in V_{pm}$.

Given an SDN $G = (V, E)$ and a request $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle$, the auxiliary graph $G'_i = (V'_i, E'_i; \omega_i)$ for request i is constructed as follows. For each switch v in V, two vertices v' and v'' are added to V'_i, and a directed edge $\langle v', v'' \rangle$ is added to E'_i. For each link $\langle u, v \rangle$ in E, an edge $\langle u'', v' \rangle$ is added to E'_i, i.e., $V'_i = \{v', v'' \mid v \in V\}$ and $E'_i = \{\langle v', v'' \rangle \mid v \in V\} \cup \{\langle u'', v' \rangle \mid \langle u, v \rangle \in E\}$. Intuitively, each edge $\langle v', v'' \rangle$ in G'_i represents switch node v and an edge $\langle u'', v' \rangle$ represents link $\langle u, v \rangle$ in G. An example of such an auxiliary graph is given in Fig. 3.

The cost model of resource usages in 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 proportion of the resource is occupied, the higher risk the resource capacity will be violated. We therefore use an exponential function to model the cost of resource usage.

Denote by $RL_{v,i}$ the residual capacity of the forwarding table at $v \in V$ and $RB_{e,i}$ the residual bandwidth of link $e = \langle v, u \rangle \in E$ when request r_i arrives. Then, the weights of their corresponding edge $e' \in E'_i$ in G'_i are

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

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 to use, since their marginal costs will increase with the increase of their utilization ratios.

Notice that the usage cost of computing resource in PMs has not been incorporated into the auxiliary graph G'_i, because admitting a request r_i via a PM-attached switch $v \in V_{pm}$ does not necessarily consume the computing resource of the PM. Only if the service chain SC_i of r_i is realized in it will the computing resource of its attached PM be consumed.

B. Algorithm

The basic idea behind the proposed algorithm is to reduce the problem in G into finding the shortest paths in a series of graphs G'_i with $1 \le i \le |S|$. In the following we first consider a single request admission. We then extend the solution to the admissions of a set of requests.

The detailed algorithm is described as follows. We first consider admitting a request $r_i \in S(t)$ where $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle$. We find a shortest path in $G'_i = (V'_i, E'_i)$ from s_i to t_i such that its corresponding routing path in G meets both its bandwidth demand b_i and its end-to-end delay d_i and there is a switch $v \in V_{pm}$ attached a PM in the path with sufficient computing resource to process its service chain SC_i. Specifically, we first remove the edges without adequate resources from G, and then construct $G'_i = (V'_i, E'_i)$ from the resulting graph G.

To include computing resource in PMs for the admission of request r_i, we then augment G'_i for each PM-attached switch $v \in V_{pm}$ and denote by $G'_{i,v} = (V'_{i,v}, E'_{i,v})$ the graph obtained by augmenting G'_i for a PM-attached switch $v \in V_{pm}$. The only difference between $G'_{i,v}$ and G'_i is that the directed edge $\langle v', v'' \rangle$ 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 shown in Fig. 4 (b). Moreover, the weight of edge $\langle v''', v'' \rangle$ is identical to 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 which usually is a constant, 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 $l(P_i(v))$ and $d(P_i(v))$ be the length and delay of $P_i(v)$, i.e., $l(P_i(v)) = \sum_{e \in P_i(v)} \omega_i(e)$ and $d(P_i(v)) = \sum_{e \in P_i(v)} d(e) + d(i, v)$, where $d(i, v)$ is the processing duration of SC_i of r_i at the PM attached to switch v.

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

We say that the derived "routing path" P_i for request r_i at step 6 in `Procedure` 1 is a pseudo-routing path or a walk, i.e., the nodes and links on P_i may appear multiple times, it can even contain cycles. This is unavoidable for a certain type of network topologies. In the following, we show the existence of a simple shortest path P_i in G for request r_i if G meets certain conditions.

Lemma 2: Given a directed weighted graph $G = (V, E)$, a specific node v, and a request r with source s and destination t, there is a simple shortest path in G from s to t that passes through node v if any path in another graph H from nodes v_0 to v does not contain any articulation points, where v_0 is a virtual node and edges $\langle v_0, s \rangle$ and $\langle t, v_0 \rangle$ are two virtual edges with weights of zeros, and they are added to graph G, i.e., $H = (V \cup \{v_0\}, E \cup \{\langle v_0, s \rangle, \langle t, v_0 \rangle\})$ is then obtained.

Procedure 1 Admitting a Single Request r_i

Input: an SDN $G = (V, E)$ and the current considering request $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle$

Output: find 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 if it exists.

1: Construct the auxiliary graph $G'_i = (V'_i, E'_i; \omega_i)$ for G;
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: Construct $G'_{i,v}$ by augmenting G'_i;
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** $(d(P_i(v)) + d(i, v) \leq d_i)$ & $(l(P_i(v)) \leq l_{\min})$ **then**
8: $P_i^{sel} \leftarrow P_i(v)$;
9: $l_{\min} \leftarrow l(P_i(v))$;
10: $v_{\min} \leftarrow v$; /* which PM will be used */
11: **end if**
12: **end for**
13: The corresponding *pseudo-routing path (walk)* P_i in G is then derived from P_i^{sel} via PM $v_{\min}$ if it exists;

Algorithm 1 A Heuristic for Admitting a Set of Requests $S(t)$

Input: an SDN $G = (V, E)$ and a set of requests $S(t)$

Output: Determine which request $r_i \in S(t)$ to be admitted and its routing path P_i^{sel}

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: Find a routing path P_i^{sel} for request r_i, by invoking `Procedure 1`;
5: **if** path P_i^{sel} does not exist **then**
6: $S' \leftarrow S' \setminus \{r_i\}$; /* remove r_i from S' */
7: **end if**
8: **end for**
9: Let r_{i_0} be the request with $l(P_{i_0}^{sel}) = \min_{r_i \in S(t)}\{l(P_i^{sel})\}$;
10: Admit request r_{i_0} using the routing path $P_{i_0}^{sel}$, and update the resource availabilities of G by deducting the resources for accommodating $P_{i_0}^{sel}$;
11: $S' \leftarrow S' \setminus \{r_{i_0}\}$.
12: **end while**

Proof: It is known that v_0 is only connected with nodes s and t in H, if there is an articulation point u in any path from v_0 to v, this implies that any path between s (or t) and node v must pass through u, thus, if a path in G from s to t must contain u, then it appears in the path at least twice. ■

Lemma 2 provides a necessary condition of the existence of a simple path in G from s to t that passes through v. That is, such a simple path exists if any path in H from v_0 to v does not contain articulation points. If for any request r and a specified node v, the condition in Lemma 2 holds, a simple shortest path in G from s to t via v can be found as follows.

We start with *the minimum-cost two edge-disjoint path problem*: Given two nodes s and t in $G(V, E)$, the problem is to find two edge-disjoint paths between s and t such that the sum of weighted edges in these two paths is minimum. There is an efficient algorithm for this problem due to Suurballe [30], and an improved algorithm later is proposed by Suurballe and Tarjan [31].

To find two edge-disjoint paths in graph G from s to t such that the cost sum of the two paths is minimum, Suurballe's algorithm proceeds as follows. It first finds a shortest path in G from s to t. It then reverses the direction of the edges in the shortest path, and finds a shortest path in the resulting graph from s to t. As a result, two edge-disjoint paths between s and t are then found through the exclusive union of the two found paths, and the cost sum of the two paths is the minimum one [30]. Clearly, this algorithm can be modified to find two node-disjoint paths between a pair of nodes so that the cost sum of the two paths is minimum, by adopting the node splitting technique [30].

We now consider the simple shortest path problem in G between a pair of nodes s and t that passes through a specified node v. We reduce this problem in G to the problem of finding two node-disjoint paths in another graph H' such that the cost sum of the two paths is minimum. We first construct a directed auxiliary graph $H' = (V_{H'}, E_{H'})$ where $V_{H'} = \{v', v'' \mid v \in V\} \cup \{v_0\}$ and $E_{H'} = \{\langle u', v'' \rangle, \langle v', u'' \rangle \mid (u, v) \in E\} \cup \{\langle v_0, s \rangle, \langle v_0, t \rangle\}$, by adding a virtual node v_0 and virtual edges into H' and assigned both newly added virtual edges $\langle v_0, s \rangle$ and $\langle v_0, t \rangle$ with weights of zeros. We then find two edge-disjoint paths in H' between v_0 and v' such that the weighted sum of the paths is minimum. We finally have a simple path in G from s to t via v that is derived from the found two node-disjoint paths, by removing the virtual node v_0 and its incident two edges. The resulting path between s and t is a simple path via v and the sum of its weighted edges is the minimum one.

Having considered a single request admission, in the following we deal with the admissions of a set of requests $S(t)$ at time slot t, by admitting the requests one by one until no more requests can be admitted. A non-admitted request in $S(t)$ is admitted immediately if it has the minimum implementation cost at that moment. Specifically, given a to-be-admitted request $r_i \in S(t)$, `Procedure 1` is employed to find a routing path for r_i *without committing the admission* which means that the SDN controller does not allocate resources to meet the demands by this request. A found path P_i^{sel} with the minimum cost among all remaining requests in $S(t)$ will be admitted and its demanded resources will be allocated to it, the residual resource availabilities in G are updated accordingly. Meanwhile, if `Procedure 1` fails to find a path P_i^{sel} for r_i, request r_i will be rejected at time slot t. 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

In the following, we first show that the solution delivered by `Algorithm 1` is feasible, and then analyze its time complexity.

Lemma 3: Given the augmented auxiliary graph $G'_{i,v} = (V'_{i,v}, E'_{i,v})$ derived from $G = (V, E)$ and a switch $v \in V_{pm}$ for request $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle \in S(t)$, the concatenation of a shortest path from s'_i to v''' and another shortest path in $G'_{i,v}$ from v''' to t'_i will result in a valid pseudo-routing path in G from s_i to t_i with PM-attached switch v in the path. Alternatively, a simple shortest path from s'_i to t'_i through v''' delivered by applying Suurballe's algorithm is also a feasible solution if G meets the condition in Lemma 2.

Proof: Let $P_i(v)$ be the concatenation of a shortest path from s'_i to v''' and a shortest path from v''' to t'_i in $G'_{i,v}$. For simplicity, we use a *link-derived edge* to represent an edge $\langle u'', v' \rangle$ in $E'_{i,v}$ that is derived from edge $\langle u, v \rangle$ in E, and a *switch-derived edge* to denote an edge $\langle v', v'' \rangle$ in $E'_{i,v}$ that is derived from switch $v \in V$. We claim that (i) path $P_i(v)$ consists of link-derived and switch-derived edges alternatively; and (ii) path $P_i(v)$ can satisfy the requirements of request r_i, i.e., the bandwidth demand b_i, the forwarding table demand, the computing resource demand for its service chain SC_i, and the end-to-end requirement d_i. Claim (i) is obvious because there is only an outgoing edge for each switch v', i.e., $\langle v', v'' \rangle$. Claim (ii) holds because the augmented auxiliary graph $G'_{i,v}$ is the result of removing the edges and switches in G'_i that cannot meet resource requirements of request r_i, and path $P_i(v)$ is feasible only when its end-to-end delay is no greater than d_i.

The feasibility of the simple shortest path via a data center if it does exist can be proven similarly, omitted. ∎

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 with attached PMs, 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.

Proof: The solution delivered by `Algorithm 1` is feasible because each auxiliary graph is constructed from the subgraph of G that only includes the resources with sufficient residual capacities for the request. Consequently, the routing path in G derived from the found path in G'_i is feasible.

The time complexity of `Algorithm 1` is analyzed as follows. In `Procedure 1`, the construction and augmentation of the auxiliary graph G'_i take $O(|V| + |E|)$ time, while finding a shortest path in each of the $|V_{pm}|$ augmented auxiliary graphs $G'_{i,v}$ 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 is $O(|S(t)|)$. If we make use of Suurballe's algorithm to find a simple shortest path from s'_i to t'_i in $G'_{i,v}$ through v''', it takes $O(|E'_i| + |V'_i| \log |V'_i|) = O(|E| + |V| \log |V|)$ time as the construction of the auxiliary graph H' and finding shortest paths in H take no more than that amount of time. 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 empirically, which can be seen in later experimental evaluations, its running time is quite high and may fail to respond to user requests on time, considering user requests arrive one by one without the knowledge of future request arrivals. In this section we devise a faster heuristic that strives for the non-trivial trade-off between the accuracy of a solution and the running time of obtaining the solution.

A. Overview

A key ingredient of this faster heuristic is that a candidate solution to admit a subset of $S(t)$ of requests is based on the residual resource capacities of G in the beginning of time slot t, and there is no updating to these residual capacities when all requests in $S(t)$ are being considered. Thus, a candidate solution is identified first. It then further refines the candidate solution iteratively until no resource capacity violation occurs.

B. Algorithm

We first find a set of candidate routing paths $\mathcal{P}_i$ in G for each request $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle \in S(t)$, without considering the resource capacity constraints of G, where a shortest path from s_i to t_i is treated as a candidate path of r_i as long as it has one PM-attached switch in V_{pm} that satisfies b_i, SC_i, and d_i. As the service chain SC_i of r_i must be served at one of $|V_{pm}|$ PM-attached switches, we can find at most $|V_{pm}|$ candidate shortest paths for each r_i. Notice that we find candidate routing paths for requests in $S(t)$ on the augmented auxiliary graphs based on the resource availability of 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 $G'_{i,v}$ 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 $\mathcal{P}_i$ be the set of candidate paths for request r_i, we then have,

$$\mathcal{P}_i = \{P_i(v_j) \mid v_j \in V_{pm}\}. \tag{2}$$

Having the set of candidate paths $\mathcal{P}_i$ for each request r_i, we then pick only one candidate path $P_i(v_j)$ in $\mathcal{P}_i$ for request r_i in a way such that the cost sum of the selected paths for all requests is minimized, while ensuring that the computing capacity of each PM is not violated. In essence, selecting a path $P_i(v_j) \in \mathcal{P}_i$ to route request $r_i \in S(t)$ is equivalent to selecting a PM attached to a switch $v \in V_{pm}$ to implement SC_i for r_i. As different PMs may have different computing capacities, this means that the processing SC_i of r_i at different PMs will incur different computing resource demands. We thus reduce this problem to the GAP, which is defined as follows. Given a set of items I and a set of bins $\mathcal{B}$, where each bin $b \in \mathcal{B}$ has a capacity $cap(b)$, each item $i \in I$ has a size $size(i, b)$, and a profit $profit(i, b)$ if item i is placed in bin b, 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.

We now 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., $cap(v_j) = LC_{v_j}$,

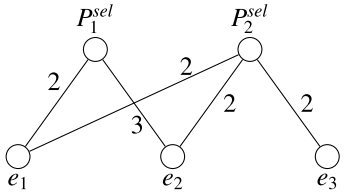


Fig. 5. An example of a bipartite graph G_b.

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.* [7] 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 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 the 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. Now, for each request allocated to a bin, its computing demand can be met without violating the computing capacity of the bin. Some requests however may violate the bandwidth and forwarding table size capacities of some links and nodes while routing their traffic. 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 $v_j \in V_{pm}$. The basic idea behind the adjustment here is to carefully find such a path with resource capacity violations iteratively and remove its request from admission. This procedure continues until there is no violation of resource capacity. To this end, a bipartite graph $G_b = (U_b, V_b, E_b)$ is constructed, where U_b is the set of selected routing paths for all potentially admitted requests, V_b is the set of edges in $\cup_{r_i \in S(t)} E(P_i^{sel})$ which each corresponds to a resource in G. There is an edge between a node $P_i^{sel} \in U_b$ and a node $e \in V_b$ if e is in P_i^{sel}. The weight of edge $(P_i^{sel}, e) \in E_b$ 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. An example of such a bipartite graph G_b is shown in Fig. 5.

To eliminate resource capacity violations, we iteratively remove one node P_i^{sel} and its incident edges in G_b with the maximum weighted sum of the incident edges, 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}. For example, in Fig. 5, both P_1^{sel} and P_2^{sel} violate the computing capacity constraints of e_1, e_2, and e_3. Since P_2^{sel} results in more violations of resource capacity constraints than P_1^{sel} does, it will be removed first. This procedure continues until no edge is left in E_b, a feasible solution will be obtained ultimately. The detailed description is given in `Algorithm` 2.

Algorithm 2 A Faster Heuristic for Routing a Set of Requests $S(t)$ Into a 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
        Find a path P_i(v_j) for r_i via node v_j, by invoking Procedure 1;
        if P_i(v_j) exists then
            P_i ← P_i ∪ {P_i(v_j)};
        end if
    end for
    if P_i is empty then
        Reject request r_i;
    else
        P ← P ∪ {P_i};
    end if
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 by invoking the algorithm in [7];
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
    Update G_b by the removal of such a node in U_b that has the maximum weighted sum of its incident edges and its incident edges from E_b.
end while
```

C. Algorithm Analysis

In the following, we show that the solution delivered by `Algorithm` 2 is a feasible solution. We then analyze the time complexity of the proposed algorithm.

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

Proof: Recall that `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

Algorithm 3 Online Algorithm Within a Finite Time Horizon T

Input: an SDN $G = (V, E)$ and time horizon T

Output: determine which request $r_i \in S(t)$ to be admitted and its routing path P_i^{sel} at each time slot t with $1 \leq t \leq T$.

1: **for** $t \leftarrow 1$ to T **do**
2: Release all resources occupied by the requests that left in the end of time slot $(t-1)$, and recalculate the residual resources in G;
3: Let $S(t)$ be the set of arrived requests in the beginning of time slot t;
4: **if** $S(t) \neq \emptyset$ **then**
5: Find a subset $S'(t) \subseteq S(t)$ of requests that are admissible at time slot t, by invoking either `Algorithm 1` or `Algorithm 2` based on the available resources in G.
6: **end if**
7: **end for**

computing capacities of nodes in V_{pm}; and (iii) eliminate the requests that violate bandwidth or forwarding table capacities. The feasibility of the solution delivered by `Algorithm 2` immediately follows from Phase (ii).

The rest is to analyze the time complexity of `Algorithm 2`. 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. The running time of Phase (ii) is dominated by the time required to solve the GAP, which is $O(|V| \cdot \frac{|S(t)|^3}{\epsilon})$ [7]. Phase (iii) takes $O(|S(t)|(|V| + |E|))$ time, there are $O(|S(t)|(|V| + |E|))$ edges in the bipartite graph G_b, following the construction of the bipartite graph. In the worst scenario, each request violates the resource capacities on all switches and links. The theorem thus holds. ■

VII. Online Algorithm

In this section, we study the online network throughput maximization problem, by considering dynamic admissions of user requests within a finite time horizon T. We will make use of the proposed algorithms in the previous section to solve this problem. We assume that the system evolves over time. The time is partitioned into equal time slots, and the user request admission scheduling proceeds in the beginning of each time slot. Some implementing requests may also leave the system, and the resources occupied by them will be released back to the system in the end of the current time slot. The released resources will be available in the beginning of the next time slot. The detailed online algorithm for dynamic request admissions is given in `Algorithm 3`.

Theorem 3: 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 with each attaching with a PM, and a finite time horizon T, there is an algorithm for the online network throughput maximization problem, `Algorithm 3`, which delivers a feasible solution in $O(\sum_{t=1}^{T}(|S(t)||V|^3 + |V| \cdot \frac{|S(t)|^3}{\epsilon}))$ time if `Algorithm 2` is used as its subroutine, where ϵ is a given constant with $0 < \epsilon \leq 1$.

Proof: The time complexity of `Algorithm 3` per time slot is the identical to the one for `Algorithm 2`, omitted. ■

VIII. Performance Evaluation

In this section, we evaluate the performance of the proposed algorithms through experimental simulations, using real and synthetic SDNs. We start with the experimental environments, we then evaluate the performance of the proposed heuristic algorithms for the network throughput maximization problem. We also evaluate the performance of the online algorithms for the online network throughput maximization problem. We finally investigate the impact of parameters on the performance of the proposed algorithms.

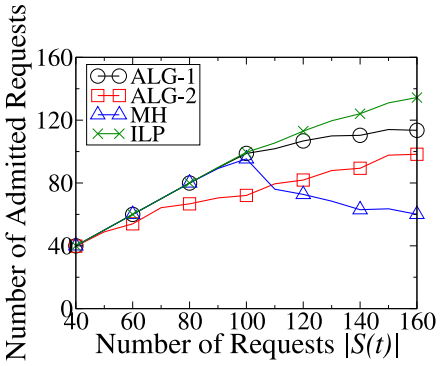
A. Experimental Environment

We adopt commonly used, real network topologies including GÉANT [23] and several ISP networks from [29] in the simulations, where GÉANT [23] is a European network consisting of 40 nodes and 122 links. The size of the forwarding table of each switch is set from 1,000 to 8,000 randomly [18]. The bandwidth of each Internet link varies from 1,000 Mbps to 10,000 Mbps [17]. There are nine PMs for the GÉANT topology as set in [12] and the number of PMs in ISP networks are provided by [26]. The computing capacity of each PM is from 4,000 to 8,000 MHz [13]. The delay of a link is between 2 milliseconds (*ms*) and 5 *ms* [17], [18]. We consider five types of middleboxes: Firewall, Proxy, NAT, IDS, and Load Balancing, and their computing demands are adopted from [12] and [23]. 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 in V are randomly drawn as the source s_i and the destination t_i of request r_i. The bandwidth demand b_i is randomly drawn from 10 to 120 Mbps [1] and the end-to-end delay d_i is set from 40 *ms* to 400 *ms* randomly [24].

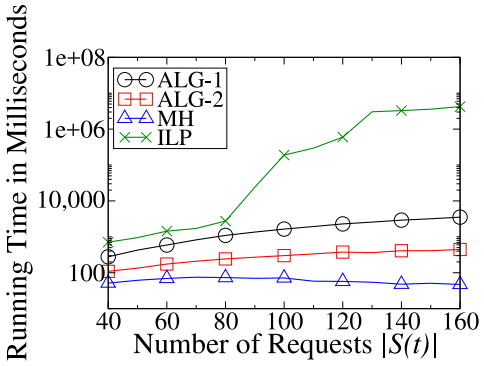
We evaluate `Algorithm 1` and `Algorithm 2` against a baseline heuristic which is described as follows. Sort all requests in $S(t)$ in increasing order of their computing resource demands, and then, for each request $r_i = \langle s_i, t_i, b_i, SC_i, d_i \rangle$ in $S(t)$, find a shortest path in G 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 t_i. We refer to this minimum-hop-based baseline as algorithm `MH`, and algorithm ILP, `Algorithm 1` and `Algorithm 2` as `ILP`, `ALG-1` and `ALG-2`, respectively. Each value in figures is the mean of the results of 30 trials.

B. Performance of Different Algorithms Within One Time Slot

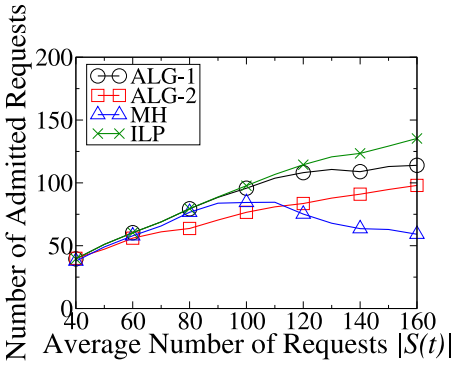
In the following, we investigate the performance of the proposed algorithms `ILP`, `ALG-1`, `ALG-2`, and `MH` in the GÉANT topology within a single time slot.



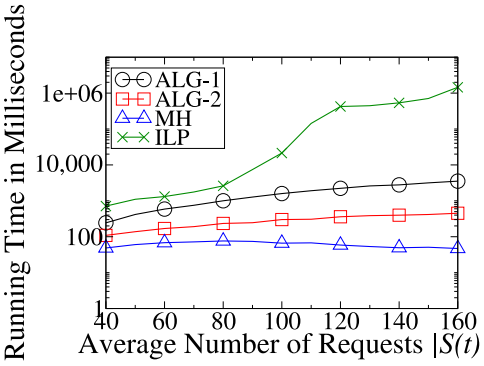
(a) The number of requests admitted if the number of requests arrived is fixed



(b) The running times (in milliseconds) on a logarithmic scale if the number of requests arrived is fixed

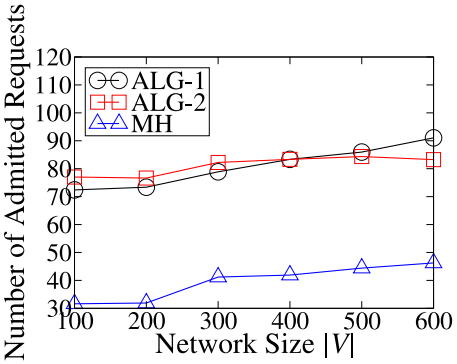


(c) The number of requests admitted if the number of requests arrived follows a Poisson distribution

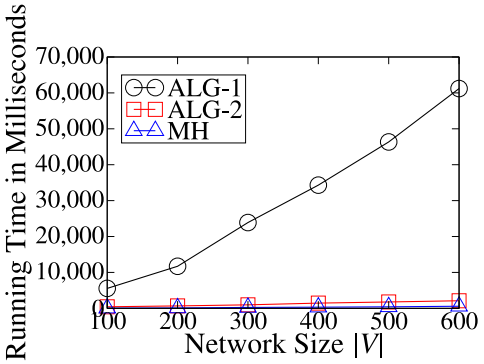


(d) The running times (in milliseconds) on a logarithmic scale if the number of requests arrived follows a Poisson distribution

Fig. 6. Performance of different algorithms on the GÉANT within one time slot.



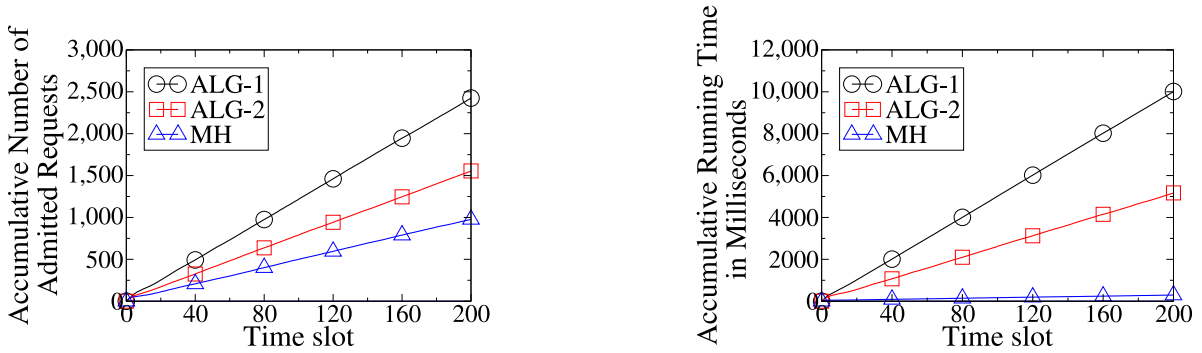
(a) The number of requests admitted by different algorithms



(b) Running time of different algorithms in milliseconds

Fig. 7. Performance of different algorithms in the GÉANT by varying the number of switches from 100 to 600, while the number of requests is fixed at 160 per time slot.

Fig. 6 (a) shows the number of requests admitted by different algorithms, when the number of requests arrived at a time slot is in the range from 40 to 160. It can be seen that both algorithms ALG-1 and MH can admit as many requests as ILP does if there are less than 100 requests. Otherwise, only algorithm ALG-1 can achieve a comparable throughput as ILP. This means that the network throughput of algorithm ALG-2 is inferior to algorithm ALG-1, and the gap between their performance enlarges from nearly zero at $|S(t)| = 40$ to 21 at $|S(t)| = 160$. The reason is that algorithm ALG-2 will reject more requests with the increase on the number of requests, as the likelihood of routing paths that algorithm ALG-2 finds for different requests being overlapping and resource violation soars. Meanwhile, it can be seen that algorithm MH outperforms algorithm ALG-2 only when the number of requests is small. Specifically, when there are 160 requests, the number of requests admitted by algorithm MH is only 60% of that by algorithm ALG-2 but runs much faster. The reason behind is that algorithm MH does not guarantee that the routing path of request r_i from its source s_i to its destination t_i



(a) Accumulative number of requests admitted

(b) Accumulative running time of algorithms

Fig. 8. Performance of different online algorithms in the GÉNT within a time horizon of 200 time slots, where the number of requests arrives at each time slot follows a Poisson distribution with a mean of 30.

has the minimum weight, since it finds shortest paths from s_i to a PM-attached switch and from that PM-attached switch to t_i separately. Fig. 6 (b) illustrates the amounts of time spent by different algorithms, from which it can be seen that the running time of algorithm ILP is several orders of magnitude of the other mentioned algorithms, while algorithm MH is the fastest one, and ALG-2 is faster than ALG-1 significantly. In addition, the running time of algorithm ILP starts rising when there are more than 80 requests. The reason is that when the number of requests is small and their resource demands are relatively small compared to the network capacity, many feasible solutions that achieves the best performance exist, yet as the resource demands become increasingly considerable relative to the network capacity, fewer optimal solutions exist, and thus ILP spends a significant amount of time on searching for such an optimal solution. Fig. 6 also demonstrates that algorithm ILP suffers poor scalability and cannot finish within a reasonable amount of time when the problem size is large.

We now evaluate the performance of different algorithms if the number of requests arrived follows a Poisson distribution with the mean between 40 and 160. The results of the four mentioned algorithms are summarized in Fig. 6 (c)-(d), from which it can be seen that the similar behavior patterns are present in Fig. 6 (a)-(b). For instance, it can be seen that the number of requests admitted by algorithm ALG-1 is identical to that by algorithm ILP when the number of requests is no more than 70. Meanwhile, the number of admitted requests by algorithm ALG-2 is on a par with that of algorithm ALG-1, and it outperforms algorithm MH by 40% when there are more than 120 requests in the set. Similarly, algorithm ILP is the slowest one while algorithm MH is the fastest one. In particular, when the mean of the number of requests is set at 140, the running time of algorithm ILP is more than 1,000 times of that of algorithm ALG-1, whilst algorithm ALG-1 is more than six times slower than algorithm ALG-2. Although the running time of algorithm MH is the fastest, the number of requests admitted by it is the smallest one.

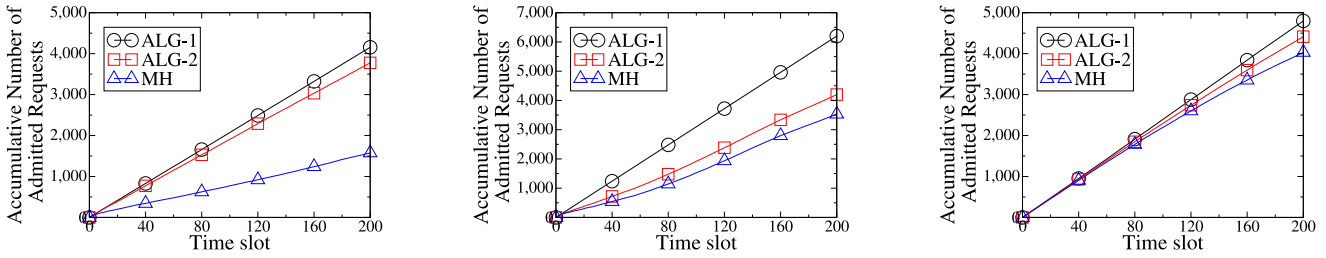
We finally evaluate the performance of different algorithms by varying the network size. As publicly available topologies such as [17] and [29] have limited sizes, we adopt the widely used Barabási-Albert model [3] to generate networks of different sizes. Namely, we vary the number of switches in an SDN from 100 to 600 while fixing the number of requests at 160. The results are depicted in Fig. 7.

It can be seen that from Fig. 7 (a) that algorithms ALG-1 and ALG-2 achieve the similar throughput, while algorithm MH admits only no more than half the requests admitted by either of the two heuristics. Fig. 7 (b) reveals that algorithm ALG-2 runs much faster than algorithm ALG-1. In contrast to high admission ratios delivered by algorithms ALG-1 and ALG-2, algorithm MH admits less than one half as many as requests as the other two algorithms, neutralizing its advantage of having the lowest running time among all three algorithms indicated in Fig. 7 (b). Fig. 7 (b) also reveals that algorithm ALG-2 runs much faster than algorithm ALG-1, e.g., algorithm ALG-1 spends 61,208 *ms* on admitting 160 requests to a network with 600 switches, while algorithm ALG-2 takes only 2,106 *ms*.

C. Performance of Online Algorithms

We now consider a time horizon that consists of 200 time slots, under which we evaluate the performance of the online versions of the proposed 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 from 1 to 10 time slots randomly.

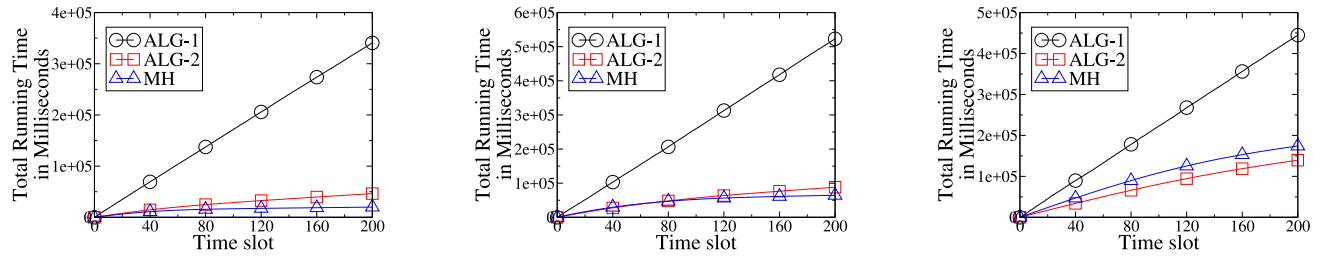
The results are summarized in Fig. 8. It can be seen from Fig. 8 (a) that algorithm MH has the lowest network throughput among the mentioned algorithms. On the other hand, algorithms ALG-1 and ALG-2 utilize resources more efficiently, and hence admit much more requests than that of algorithm MH by 150% and 50%, respectively. It can also be seen from Fig. 8 (b) that the running time of algorithm MH is negligible compared with those of algorithms ALG-1 and ALG-2. It must be noticed that this running time comes at the cost of admitting much fewer requests. Although the running time of algorithm ALG-2 is less than that of algorithm ALG-1, the gap between them becomes smaller. Specifically, algorithm ALG-2 is only half of the running time of algorithm ALG-1 while it has only 10% of the running time of algorithm ILP. The main reason is that the additional time incurred from constructing an instance of the GAP cannot be ignored when the



(a) The accumulative number of requests admitted in the AS-4755 network

(b) The accumulative number of requests admitted in the AS-1755 network

(c) The accumulative number of requests admitted in the AS-3967 network

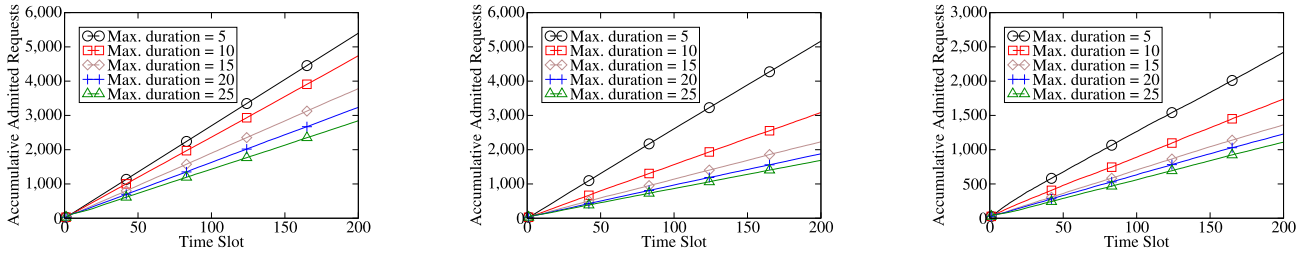


(d) The accumulative running time of different algorithms in the AS-4755 network

(e) The accumulative running time of different algorithms in the AS-1755 network

(f) The accumulative running time of different algorithms in the AS-3967 network

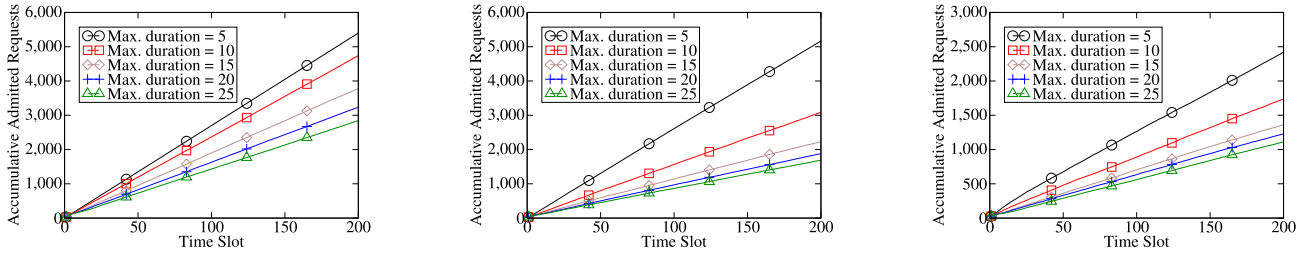
Fig. 9. The accumulative number of admitted requests and the running time of different online algorithms based on algorithms ALG-1, ALG-2 and MH for a time horizon of 200 time slots in ISP networks.



(a) The number of requests admitted by ALG-1

(b) The number of requests admitted by ALG-2

(c) The number of requests admitted by MH



(d) The number of requests admitted by ALG-1

(e) The number of requests admitted by ALG-2

(f) The number of requests admitted by MH

Fig. 10. The accumulative number of admitted requests and running time of different online algorithms based on algorithms ALG-1, ALG-2, and MH with different maximum durations of requests when the network is of the GÉANT.

number of requests is relatively small, which will be offset when the number of requests is greater.

The rest is to evaluate algorithms ALG-1, ALG-2, and MH in three network topologies from [29]: AS-4755 is a network with 121 switches and 296 links, AS-1755 with 172 switches and 762 links, and AS-3967 with 212 switches and 886 links.

The results are illustrated in Fig. 9, from which it can be seen that in terms of the performance, algorithm ALG-1 is the best while algorithm MH is the worst. The performance gap between algorithm ALG-1 and algorithm ALG-2 is small compared to that in the GÉANT, since the size of these three networks is larger than that of the GÉANT, and routing paths delivered by algorithm ALG-2 for different requests at each time slot are less likely to overlap. In AS-4755, the difference on requests admitted by algorithms ALG-1 and ALG-2 is less than 500, while algorithm MH admits no more than 40% the number of requests by the other two algorithms. On the other hand, the performance of algorithm MH in both AS-1755 and AS-3967 improves due to the larger resource capacity of the network. The accumulative running time of these three algorithms when admitting requests into different networks are

shown in Fig. 9 (d)-(f). The results in both Fig. 9 (d) and (e) are similar: the running time of algorithm MH is slightly smaller than that of algorithm ALG-2, and algorithm ALG-1 is much slower than both algorithms ALG-1 and MH. However, we notice from Fig. 9 (f) that algorithm ALG-2 is faster than algorithm MH, since algorithm ALG-2 only needs to find shortest paths in a graph once.

D. Impact of Request Durations on the Performance of Different Online Algorithms

We finally evaluate the impact of the maximum duration of requests on the performance of different online algorithms based on algorithms ALG-1, ALG-2, and MH, by varying the maximum duration from 5 time slots to 25 time slots. The results are presented in Fig. 10. From Fig. 10 (a)-(b) we can see that the longer the maximum duration, the fewer requests admitted by all mentioned algorithms, because longer durations result in less available resources in the network. Fig. 10 (a)-(c) show that when the maximum duration of requests is five time slots, algorithm ALG-2 admits as many requests as algorithm ALG-1, yet algorithm MH only admits half the number of requests as the two heuristics. We also see that an increase in the request durations has a greater impact on algorithms ALG-2 and MH than that on algorithm ALG-1, as algorithm ALG-1 is more exhaustive. In addition, it can also be seen from Fig. 10 (d)-(f) that longer durations are associated with less running time, because if other parameters keep unchanged and request durations become longer, more resources will be fully occupied and accordingly, the auxiliary graphs will have fewer vertices and edges, shortening operations on the auxiliary graphs and reducing the running time of all algorithms.

IX. Conclusion

In this paper, we studied the admissions of user requests with each having a sequence of network functions in an SDN so that the network throughput can be maximized, subject to the constraints of forwarding table capacity, network bandwidth capacity, and computing resource capacity at PMs. We first formulated an ILP solution when the problem size is small. We then devised two heuristic algorithms that strive for a fine tradeoff between the solution accuracy and the running time to obtain the solutions. We also investigated the dynamic admissions of requests within a finite time horizon by extending the proposed algorithms to solve dynamic request admissions. We finally evaluated the performance of the proposed algorithms through simulations, using real and synthetic network topologies. Experimental results demonstrated that both proposed algorithms admit more requests than a baseline algorithm, and the quality of solutions delivered is on a par with that of optimal solutions yet the proposed algorithms are significantly faster.

Acknowledgment

The authors really appreciate the three anonymous referees and the associate editor for their expertise comments and constructive suggestions, which have helped them improve the quality and presentation of the paper greatly.

References

[1] *Amazon EC2 Instance Configuration*, Amazon Web Services, Inc., Seattle, WA, USA, 2017. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-ec2-config.html

[2] J. W. Anderson, R. Braud, R. Kapoor, G. Porter, and A. Vahdat, "xOMB: Extensible open middleboxes with commodity servers," in *Proc. ACM/IEEE ANCS*, 2012, pp. 49–60.

[3] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[4] Z. Cao, M. Kodialam, and T. V. Lakshman, "Traffic steering in software defined networks: Planning and online routing," in *Proc. ACM DCC*, Chicago, IL, USA, 2014, pp. 65–70.

[5] M. Charikar, Y. Naamad, J. Rexford, and X. K. Zou. (2014). *Multi-Commodity Flow With in-Network Processing*. [Online]. Available: http://www.cs.princeton.edu/~jrex/papers/mopt14.pdf

[6] C.-H. Chi, J. Deng, and Y.-H. Lim, "Compression proxy server: Design and implementation," in *Proc. USITS*, Boulder, CO, USA, 1999, p. 10.

[7] R. Cohen, L. Katzir, and D. Raz, "An efficient approximation for the generalized assignment problem," *Inf. Process. Lett.*, vol. 100, no. 4, pp. 162–166, 2006.

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

[9] A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella, "Stratos: Virtual middleboxes as first-class entities," Dept. Comput. Sci., Univ. Wisconsin at Madison, Madison, WI, USA, Tech. Rep. TR1771, Jun. 2012.

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

[11] A. Gupta *et al.*, "SDX: A software defined Internet exchange," in *Proc. ACM SIGCOMM*, Chicago, IL, USA, 2014, pp. 551–562.

[12] A. Gushchin, A. Walid, and A. Tang, "Scalable routing in SDN-enabled networks with consolidated middleboxes," in *Proc. ACM HotMiddlebox*, London, U.K., 2015, pp. 55–60.

[13] *Servers for Enterprise—BladeSystem, Rack & Tower and Hyperscale*, Hewlett-Packard, Palo Alto, CA, USA, 2015. [Online]. Available: http://www8.hp.com/us/en/products/servers/

[14] M. Honda *et al.*, "Is it still possible to extend TCP?" in *Proc. ACM IMC*, Berlin, Germany, 2011, pp. 181–194.

[15] H. Huang, S. Guo, J. Wu, and J. Li, "Joint middlebox selection and routing for software-defined networking," in *Proc. IEEE ICC*, 2016, pp. 1–6.

[16] M. Huang, W. Liang, Z. Xu, M. Jia, and S. Guo, "Throughput maximization in software-defined networks with consolidated middleboxes," in *Proc. IEEE LCN*, 2016, pp. 298–306.

[17] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

[18] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

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

[20] Y. Li, L. T. X. Phan, and B. T. Loo, "Network functions virtualization with soft real-time guarantees," in *Proc. IEEE INFOCOM*, San Francisco, CA, USA, 2016, pp. 1–9.

[21] T. Lukovszki, M. Rost, and S. Schmid, "It's a match!: Near-optimal and incremental middlebox deployment," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 1, pp. 31–36, 2016.

[22] T. Lukovszki and S. Schmid, "Online admission control and embedding of service chains," in *Proc. SIROCCO*, 2015, pp. 104–118.

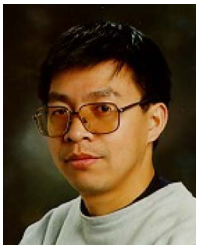
[23] J. Martins *et al.*, "ClickOS and the art of network function virtualization," in *Proc. USENIX NSDI*, Seattle, WA, USA, 2014, pp. 459–473.

[24] *Plan Network Requirements for Skype for Business*, Microsoft, Redmond, WA, USA, 2015. [Online]. Available: https://technet.microsoft.com/en-us/library/gg425841.aspx

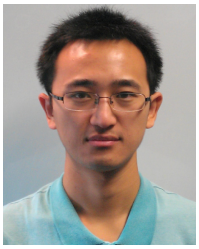
[25] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 1st Quart., 2016.

[26] Z. A. Qazi *et al.*, "SIMPLE-fying middlebox policy enforcement using SDN," in *Proc. ACM SIGCOMM*, Hong Kong, 2013, pp. 27–38.
[27] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proc. USENIX NSDI*, San Jose, CA, USA, 2012, p. 24.
[28] J. Sherry *et al.*, "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proc. ACM SIGCOMM*, Helsinki, Finland, 2012, pp. 13–24.
[29] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," in *Proc. ACM SIGCOMM*, 2002, pp. 133–145.
[30] J. W. Suurballe, "Disjoint paths in a network," *Networks*, vol. 4, no. 2, pp. 125–145, 1974.
[31] J. W. Suurballe and R. E. Tarjan, "A quick method for finding shortest pairs of disjoint paths," *Networks*, vol. 14, no. 2, pp. 325–336, 1984.
[32] Z. Xu *et al.*, "Approximation and online algorithms for NFV-enabled multicasting in SDNs," in *Proc. IEEE ICDCS*, 2017, pp. 625–634.
[33] S. Q. Zhang *et al.*, "Sector: TCAM space aware routing on SDN," in *Proc. IEEE Int. Teletraffic Congr.*, 2015, pp. 216–224.

Meitian Huang received the B.Sc. (with First Class Hons.) degree in computer science with the Australian National University in 2015. He is currently pursuing the Ph.D. degree with the Research School of Computer Science, Australian National University. His research interests include software-defined networking, algorithm design and analysis, and cloud computing.



Weifa Liang (M'99–SM'01) received the B.Sc. degree from Wuhan University, China, in 1984, the M.E. degree from the University of Science and Technology of China in 1989, and the Ph.D. degree from the Australian National University in 1998, all in computer science. He is currently a Professor with the Research School of Computer Science, Australian National University. His research interests include design and analysis of energy efficient routing protocols for wireless ad hoc and sensor networks, cloud computing, software-defined networking, design and analysis of parallel and distributed algorithms, approximation algorithms, combinatorial optimization, and graph theory.



Zichuan Xu (M'17) received the B.Sc. and M.E. degrees from the Dalian University of Technology in China in 2008 and 2011, and the Ph.D. degree from the Australian National University in 2016, all in computer science. He was a Research Associate with the Department of Electronic and Electrical Engineering, University College London, U.K. He is currently an Associate Professor with the School of Software, Dalian University of Technology, China. His research interests include cloud computing, software-defined networking, network function virtualization, wireless sensor networks, algorithmic game theory, and optimization problems.

Song Guo (M'02–SM'11) received the Ph.D. degree in computer science from the University of Ottawa, Canada, in 2005. He is currently a Full Professor with the Department of Computing, Hong Kong Polytechnic University, Hong Kong. His research interests are mainly in the areas of protocol design and performance analysis for wireless networks and distributed systems. He has published over 250 papers in refereed journals and conferences in these areas and was a recipient of three IEEE/ACM Best Paper Awards. He currently serves as an Associate Editor of the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, an Associate Editor of the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING for the track of Computational Networks, and on editorial boards of many others. He has also been in organizing and technical committees of numerous international conferences. He is a Senior Member of the ACM.