SDN-Based Traffic Aware Placement of NFV Middleboxes

Wenrui Ma, *Student Member, IEEE*, Jonathan Beltran, Zhenglin Pan, Deng Pan, and Niki Pissinou

***Abstract*—Network function virtualization (NFV) enables flexible deployment of middleboxes as virtual machines running on general hardware. Since different middleboxes may change the volume of processed traffic in different ways, improper deployment of NFV middleboxes will result in hot spots and congestion. In this paper, we study the traffic changing effects of middleboxes, and propose software-defined networking based middlebox placement solutions to achieve optimal load balancing. We formulate the traffic aware middlebox placement (TAMP) problem as a graph optimization problem with the objective to minimize the maximum link load ratio. First, we solve the TAMP problem when the flow paths are predetermined, such as the case in a tree. For a single flow, we propose the least-first-greatest-last (LFGL) rule and prove its optimality; for multiple flows, we first show the NP-hardness of the problem, and then propose an efficient heuristic. Next, for the general TAMP problem without predetermined flow paths, we prove that it is NP-hard even for a single flow, and propose the LFGL based MinMax routing algorithm by integrating LFGL with MinMax routing. We use a joint emulation and simulation approach to evaluate the proposed solutions, and present extensive experimental and simulation results to demonstrate the effectiveness of our design.**

***Index Terms*—Network function virtualization, software-defined networking, middlebox.**

I. Introduction

THE ADVANCEMENT of virtualization technology [16] has made Network Function Virtualization (NFV) [29] a promising architecture for middleboxes. Middleboxes are traffic processing appliances that are widely deployed in data centers, enterprise networks, and telecommunications networks [21]. Traditional middleboxes are proprietary hardware devices that implement specialized functions such as firewalls, VPN proxies, and WAN optimizers [27]. Such hardware middleboxes suffer from a number of drawbacks [40], [48], including high cost, short life time, function inflexibility, and difficulty to scale up.

NFV decouples network functions from physical equipment, and implements middleboxes by running network function software on virtualized general hardware [17]. An NFV server is an industry standard server that hosts multiple virtual machines (VMs), each implementing a middlebox function with specialized software programs. Software middleboxes can be instantiated at, or moved to, general servers at various locations of the network, without the need to install new hardware. Benefiting from the underlying virtualization technology, NFV enjoys many advantages not available in traditional hardware middleboxes [5], such as fast deployment, reduced energy consumption, and real-time optimization.

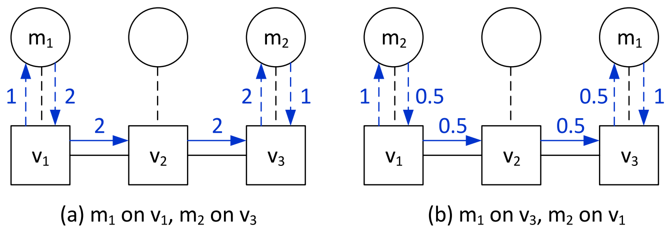


Fig. 1. Traffic changing effects of middleboxes.

Unlike switches or routers that are only forwarding traffic, most middleboxes are traffic processing devices, and may change the volume of processed traffic and may do it in different ways. For example, the Citrix CloudBridge WAN optimizer [1] may compress traffic to 20% of its original volume before sends it to the next hop. On the other hand, a Stateless Transport Tunneling (STT) proxy [2] adds 76 bytes to each processed packet due to the encapsulation overhead. Finally, a firewall will keep the traffic rates of allowed flows unchanged and reduce the rates of denied flows to zero.

The following toy example in Fig. 1 illustrates the traffic changing effects of middleboxes. Consider a network consisting of three nodes v_1, v_2 and v_3, and two links (v_1, v_2) and (v_2, v_3). Each node has an attached NFV server, and each server can host a single middlebox. A flow f starts at v_1 and ends at v_3, whose initial traffic rate is 1. Two middleboxes m_1 and m_2 need to be applied to f. m_1 will double the traffic rate, while m_2 will cut the traffic rate in half. If install m_1 on v_1 and m_2 on v_3, the load of links (v_1, v_2) and (v_2, v_3) will be $1 \times 2 = 2$, as shown in Fig. 1(a). However, by installing m_1 on v_3 and m_2 on v_1, we can reduce the load of both links to $1 \times 0.5 = 0.5$, as shown in Fig. 1(b).

As can be seen, the flexibility of VMs brings a couple of challenges for efficient NFV implementation. First, since there may exist multiple candidate NFV servers, a strategic deployment plan is necessary to determine the optimal location for a middlebox. Next, due to traffic changing effects, the order to deploy different types of middleboxes is critical for balancing traffic load in the network.

Manuscript received February 20, 2017; revised June 5, 2017; accepted June 27, 2017. Date of publication July 19, 2017; date of current version September 7, 2017. The associate editor coordinating the review of this paper and approving it for publication was G. Leduc. *(Corresponding author: Deng Pan.)*

The authors are with the School of Computing and Information Sciences, Florida International University, Miami, FL 33199 USA (e-mail: wma006@fiu.edu; jbelt021@fiu.edu; zpan004@fiu.edu; pand@fiu.edu; pissinou@fiu.edu).

Digital Object Identifier 10.1109/TNSM.2017.2729506

In this paper, we study optimal deployment of NFV middleboxes with the objective to achieve load balancing, and will focus on the persistent and large-sized elephant flows [20] but not the transient and small-sized mice flows [20], for the following three reasons. First, since elephant flows constitute a majority of the network traffic [42], optimizing elephant flows will efficiently help balance the entire network. Second, mice flows are transient, and may leave the network before the calculated optimization scheme takes effect, making it difficult to achieve the optimization objective. Third, the number of mice flows is much greater than that of elephant flows [20], and the computation cost to manage so many dynamic flows is prohibitive. Therefore, our design is to deploy independent middleboxes for each elephant flow to avoid resource contention and congestion, but instead let mice flows utilize the leftover processing capacity of middleboxes that have been deployed for elephant flows. The solution for mice flows will not be the focus of this paper.

The solution proposed in this paper leverages the emerging Software-Defined Networking (SDN) architecture [8], which enables efficient optimization by decoupling the network control plane and data plane. An SDN based prototype has been implemented to demonstrate the practicality of our design.

Our main contributions are summarized as follows. First, we formulate the Traffic Aware Middlebox Deployment (TAMP) problem as a graph optimization problem with the objective to minimize the maximum link load ratio in the network. Second, when flow paths are predetermined, such as the case in the tree topology, we propose the Least-First-Greatest-Last (LFGL) rule to place middleboxes for a single flow, and prove its optimality. For multiple flows, we show that the TAMP problem is NP-hard by reduction from the 3-Satisfiability problem, and propose an efficient heuristic. Third, for the general scenario without predetermined flow paths, we prove that the TAMP problem is NP-hard even for a single flow by reduction from the Hamiltonian Cycle problem, and propose the LFGL based MinMax routing algorithm that integrates LFGL with MinMax routing. Fourth, we have implemented the proposed algorithms in a prototype with the open-source SDN controller Floodlight and network emulator Mininet. Finally, we present extensive experimental and simulation results to demonstrate the effectiveness of the proposed algorithms.

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

II. Related Works

NFV has recently attracted significant attention from both industry and academia as an important shift in network function provisioning. A wide range of research has been conducted on the NFV architecture, software platform, middlebox placement, and resource management, as outlined below. However, to the best of our knowledge, traffic aware deployment of NFV middleboxes, and in particular the traffic changing effects, have not been well investigated in the literature.

NFV architecture: Sekar *et al.* [46] propose an NFV architecture named CoMb that explores consolidation opportunities to reduce network provisioning cost and load skew. Anderson *et al.* [14] propose the xOMB architecture for building scalable and extensible middleboxes. ServerSwitch [37] is a low cost architecture that integrates a powerful multi-core commodity server with a programmable switching chip. PacketShader [30] is a software router framework with Graphics Processing Unit (GPU) acceleration that brings significantly higher throughput over previous CPU-only implementation. Egi *et al.* [23] identify principles for constructing high-performance software router systems on commodity hardware, and show that the solutions based on current and near-future commodity hardware are flexible, practical, and inexpensive. This work can benefit from those research advances by implementing our solutions based on the above NFV architectures.

Software platform: Hwang *et al.* [32] propose the NetVM network virtualization platform, built on top of the KVM infrastructure and Intel DPDK library, to dynamically scale, deploy, and reprogram network functions. By extending the idea of Click [36], ClickOS [39] develops a high-performance, virtualized software middlebox platform, which enables hundreds of concurrent virtual network functions without introducing significant overhead in packet processing. The above results can be used as the software foundation to efficiently implement the algorithms proposed in this work.

Middlebox placement: SIMPLE [44] presents a SDN-based policy enforcement layer for efficient middlebox-specific traffic steering. Although SIMPLE is designed with the constraints of legacy hardware middleboxes and existing SDN interfaces, it can also be applied to software middleboxes running on commodity hardware. The header modifications induced by middleboxes make it difficult to reason about the correctness of network-wide policy enforcement. To address this challenge, Fayazbakhsh *et al.* [24] propose an extended SDN architecture called FlowTags, in which middleboxes add packet tags to provide the necessary context for systematic policy enforcement. Our work differs from the above ones by focusing on middlebox placement with the objective to achieve load balancing.

Switch memory management: To efficiently manage the expensive and power-hungry ternary content-addressable memory (TCAM) in switches, multiple works [26], [45], [49] have studied shrinking the routing table size by aggregating rules. Uzmi *et al.* [47] further propose a practical and near-optimal aggregation scheme to minimize the switch table size. Katta *et al.* [35] propose the CacheFlow system for SDN to cache the most popular rules in a small TCAM, while relying on software to handle the cache miss traffic. Kang *et al.* [43] propose a rule placement algorithm to distribute forwarding policies across general SDN networks while managing rule-space constraints. Our work relates to the above ones by also considering the limit of middleboxes that can be hosted at a node due to resource constraints, such as switch TCAM or NFV server memory.

III. Problem Formulation

In this section, we formulate the Traffic Aware Middlebox Placement (TAMP) problem.

Consider a network represented by a directed graph $G = (V, E)$. Each node $v \in V$ may have an attached NFV server, and its space capacity is denoted as $sc_v \geq 0$, i.e., the maximum number of middleboxes to host. For simplicity, we assume that each middlebox needs one space, and more processing power can be achieved by additional middlebox instances. A link $(u, v) \in E$ has a bandwidth capacity $bc_{u,v} \geq 0$, i.e., the available bandwidth. Its current link load is denoted as $l_{u,v}$.

Use M to denote the complete set of middlebox types. Each middlebox type $m \in M$ has an associated traffic changing factor $alter_m$, where $1 + alter_m$ is the ratio of the traffic rate of a flow after and before being processed by m.

Let F denote the set of flows. Each flow $f \in F$ is represented as a 4-tuple (s_f, d_f, t_f, M_f), in which $s_f \in V$ is the source node, $d_f \in V$ is the destination node, t_f is the initial traffic rate at the ingress point, and $M_f \subseteq M$ is the set of required middleboxes.

When a flow f enters the network, a path $route_f$ will be assigned for the flow, which is a decision variable defined as

$$route_f(u, v) = \begin{cases} 1, & \text{if flow } f \text{ traverses link } (u, v). \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

To avoid performance degradation for TCP flows, a flow is not allowed to be split among multiple paths [31].

Use t_f^{v-} and t_f^{v+} to denote the traffic rate of flow f before entering and after leaving node v, respectively. If f is processed by a middlebox of type m, or middlebox m for short, at v, then $t_f^{v+} = t_f^{v-}(1 + alter_m)$. Note that $t_f^{s_f-} = t_f$. For convenience, use $t_f^{u,v} = t_f^{u+} = t_f^{v-}$ to represent the traffic rate of f on its path link (u, v).

In addition, a placement scheme $place_f$ will determine the locations to install each middlebox $m \in M_f$, which is a decision variable defined as

$$place_f(m, v) = \begin{cases} 1, & \text{if middlebox } m \text{ is installed at node } v. \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

Define the ratio between the aggregate load and capacity of a link to be the link load ratio, which is also called traffic intensity in queuing theory [34] and determines the queuing delay. To achieve load balancing, our objective is to minimize the maximum link load ratio in the network by optimizing $route_f$ and $place_f$ for each flow $f \in F$, as shown Equation (3). Our solutions can also easily adapt to other optimization objectives [15], [28], such as minimizing the path cost or maximizing the residual capacity.

$$\textbf{minimize } maxRatio \quad (3)$$

subject to the following constraints:

$$\forall (u, v) \in E:$$
$$\frac{l_{u,v} + \sum_{f \in F} route_f(u, v) t_f^{u,v}}{bc_{u,v}} \leq maxRatio \quad (4)$$
$$\forall f \in F:$$
$$\sum_{u \in V} route_f(s_f, u) = \sum_{u \in V} route_f(u, s_f) + 1, \quad (5)$$
$$\sum_{u \in V} route_f(u, d_f) = \sum_{u \in V} route_f(d_f, u) + 1 \quad (6)$$
$$\forall f \in F, \forall v \in V - \{s_f, d_f\}:$$
$$\sum_{u \in V} route_f(u, v) = \sum_{u \in V} route_f(v, u) \quad (7)$$
$$\forall v \in V:$$
$$\sum_{f \in F} \sum_{m \in M_f} place_f(m, v) \leq sc_v \quad (8)$$
$$\forall f \in F, \forall m \in M_f, \forall v \in V:$$
$$place_f(m, v) \leq \sum_{u \in V} (route_f(u, v) + route_f(v, u)) \quad (9)$$
$$\forall f \in F, \forall m \in M_f:$$
$$\sum_{u \in V} place_f(m, u) = 1 \quad (10)$$
$$\forall f \in F, \forall (u, v) \in E:$$
$$t_f^{u,v} =$$
$$t_f^{u-} route_f(u, v) \prod_{m \in M_f} (1 + place_f(m, u) alter_m) \quad (11)$$

Equation (4) states that, for a link (u, v), its load ratio $(l_{u,v} + \sum_{f \in F} route_f(u, v) t_f^{u,v})/bc_{u,v}$ should be less than or equal to the optimization objective $maxRatio$. Equations (5) and (6) enforce each flow f to start and end at its source s_f and destination d_f, respectively. Equation (7) guarantees flow conservation at each intermediate node on the path, i.e., no flow generation or termination at an intermediate node. Equation (8) states that, for a node v, the total space demand of hosted middleboxes $\sum_{f \in F} \sum_{m \in M_f} place_f(m, v)$ should not exceed its space capacity sc_v. Equation (9) states that a middlebox m can be installed only on a node v that the flow path traverses. Equation (10) states that a middlebox m should be installed once and only once. Equation (11) states that, for a flow f, its traffic rate on a link (u, v) of its path, i.e., $route_f(u, v) = 1$, or its traffic rate when leaving u, or its traffic rate when entering v, is equal to its traffic rate when entering u, i.e., t_f^{u-}, multiplying the traffic changing ratios $1 + place_f(m, u) alter_m$ of all the middleboxes m placed at node u.

It can be seen that, optimal performance can be achieved by minimizing the maximum link load ratio on the routing paths of the flows in F, although the objective $maxRatio$ is the maximum link load ratio of the entire network. Proofs are omitted. Thus, the following proposed solutions will focus on minimizing the maximum link load ratio on flow paths.

IV. Traffic Aware Middlebox Placement With Predetermined Paths

In this section, we solve the TAMP problem when the flow paths, i.e., $route$, have been determined, or are unique in certain network topologies, such as the popular tree topology. We start with a single flow, i.e., $|F| = 1$, and propose the Least-First-Greatest-Last (LFGL) rule to achieve optimal performance. When there are multiple flows, i.e., $|F| > 1$, we prove that the TAMP problem is NP-hard by reduction from the 3-Satifiability problem, and propose an efficient heuristic.

A. Middlebox Placement for Single Flow

In reality, flows tend not to arrive at exactly the same time. Even if multiple flows arrive simultaneously in a software-defined network (SDN), the central controller will have to process them one by one. Thus, solutions for a single flow are of practical importance, especially for SDNs. Assume that the flow set F has only a single flow f, and the path $route_f$ has been determined. Obviously, a valid placement solution $place_f$ exists, if and only if the path links have sufficient bandwidth capacity, and the number of available NFV spaces on the routing path is greater than or equal to the number of required middleboxes, i.e., $|M_f|$.

The basic idea of our solution is to push heavy traffic out of the network core by decreasing the traffic rate at the beginning of the path, and increasing it at the end of the path. Based on this observation, we propose an efficient rule called *Least-First-Greatest-Last* (LFGL) to optimally place middleboxes. The rule starts by sorting all the middleboxes $m \in M_f$ based on their traffic changing factors $alter_m$. It then places the middleboxes with non-positive factors, or shrinking middleboxes, from the head of the path in an increasing order. When a node has no space left, LFGL continues with the next node on the path. After finishing placing shrinking middleboxes, the rule switches to middleboxes with positive traffic changing factors, or expanding middleboxes, and place them from the path tail in the decreasing order of their factors. The deployment succeeds if all middleboxes are placed, and fails otherwise.

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

Theorem 1: The Least-First-Greatest-Last rule minimizes the maximum link load ratio on the flow path.

Proof: For the purpose of contradiction, assume that a different placement scheme $place'_f$ achieves maximum link load $maxRatio'$ lower than that of LFGL, i.e., $maxRatio' < maxRatio$.

Without loss of generality, assume that the differences between the two placement schemes include shrinking middleboxes, and m is the one with the least traffic changing factor. By the LFGL rule, m is installed in the first available node u at its placement time, i.e., $place_f(m, u) = 1$. By comparison, the other placement scheme installed m on a different node u', i.e., $place'_f(m, u') = 1$, which must be after u on the flow path, and instead a different middlebox m' is placed on u, i.e., $place'_f(m', u) = 1$. Apparently, the placement of m' is also different in $place_f$ and $place'_f$. Since among the differences between $place_f$ and $place'_f$, m has the least traffic changing factor, we know that $alter_m \le alter_{m'}$.

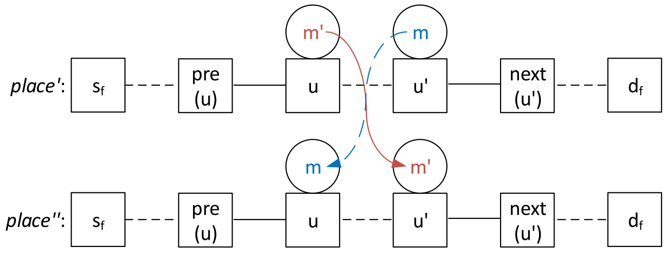


Fig. 2. Proof of Theorem 1.

Next, as shown in Fig. 2, we create a new placement scheme $place''_f$ by switching the locations of m and m' in $place'_f$, i.e.,

$$place''_f(n, v) = \begin{cases} place'_f(n, v), & \text{if } n \neq m, n \neq m' \\ place'_f(m, u')(= 1), & \text{if } n = m, v = u \\ place'_f(m', u)(= 1), & \text{if } n = m', \ v = u' \end{cases} \tag{12}$$

Then, the maximum link load ratio $maxRatio''$ of the new placement $place''_f$ will be less than or equal to that of $place'_f$, i.e., $maxRatio'' \le maxRatio'$. Denote the traffic rates of f on link $(v, w) \in E$ under $place'_f$ and $place''_f$ as $t'^{v,w}_f$ and $t''^{v,w}_f$, respectively. Analyze the following three types of links.

1) For a link (v, w) between s_f and u, since the middleboxes placed before u are the same under $place'_f$ and $place''_f$, the traffic rates of f on such a link are also the same under both schemes, i.e., $t''^{v,w}_f = t'^{v,w}_f$.
2) For a link (v, w) between u and $next_f(u')$, as the locations of m and m' are exchanged in $place''_f$, $t''_f(v, w) = t'_f(v, w)(1+alter_m)/(1+alter_{m'})$. Since $alter_m \le alter_{m'}$ as shown above, we know $t''^{v,w}_f \le t'^{v,w}_f$.
3) For a link (v, w) between $next_f(u')$ and d_f, since the middleboxes placed after u' are the same under $place'_f$ and $place''_f$, the traffic rates of f on such a link are the same, i.e., $t''^{v,w}_f = t'^{v,w}_f$.

To sum up, for each link on the flow path of f, $place''_f$ achieves a lower or equal traffic rate for the flow than $place'_f$, resulting in a lower or equal maximum link load ratio, and it has one less difference with $place_f$ generated by LFGL. Continuing this process and eliminating all the differences between $place'_f$ and $place_f$, it can be shown by induction that $place_f$ achieves no higher maximum link load ratio than that of $place'_f$, i.e., $maxRatio \le maxRatio'$, which contradicts the assumption. ■

B. Middlebox Placement for Multiple Flows

We now solve the problem to place middleboxes for multiple flows with predetermined paths. As explained in the introduction, we do not let different elephant flows share the same middlebox to avoid hot spots, but instead the leftover processing capacity of installed middleboxes will be utilized by mice flows.

Theorem 2: The Traffic Aware Middlebox Placement placement problem for multiple flows with predetermined paths is NP-hard.

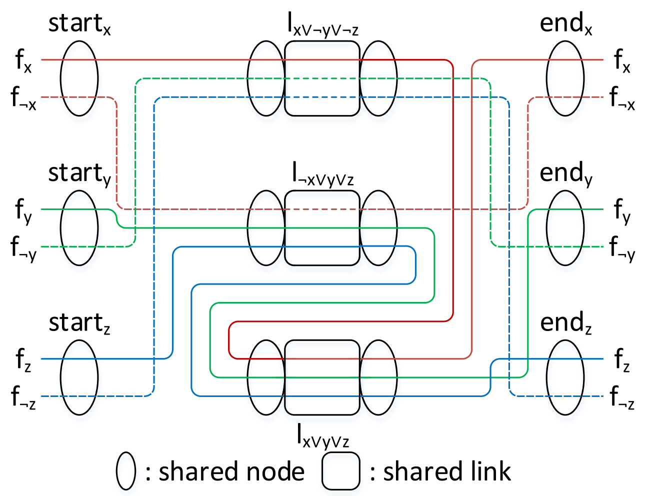


Fig. 3. Reduction from 3-Satisfiability to TAMP for multiple flows with predetermined paths.

Proof: We prove by reduction from the 3-Satisfiability problem. The 3-Satisfiability problem decides whether a boolean formula in 3-CNF, i.e., the conjunction normal form with three boolean variables per clause, is satisfiable. An example is $(x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (x \vee y \vee z)$.

The reduction process is as follows. For a pair of boolean variables x and $\neg x$, we create two corresponding flows f_x and $f_{\neg x}$, and two nodes $start_x$ and end_x, each with one available middelbox space. Each flow f has an initial traffic rate of $t_f = 1$, and needs a single middlebox $M_f = \{m\}$ with $alter_m = -1 + \epsilon$, where ϵ is a small positive quantity less than one. In other words, the middlebox m will change the traffic rate of the processed flow to $1 \times (1 + alter_m) = \epsilon$. The two flows f_x and $f_{\neg x}$ both start at the shared node $start_x$, i.e., $s_{f_x} = s_{f_{\neg x}} = start_x$, and end at end_x, i.e., $d_{f_x} = d_{f_{\neg x}} = end_x$. For each clause $C = x_1 \vee x_2 \vee x_3$, we create a shared link l_C with a bandwidth capacity of 10, which will be traversed by the three flows corresponding to x_1, x_2, and x_3. When a boolean variable is included in multiple clauses, its corresponding flow will traverse multiple links one by one. Except the shared links, different flows have separate links for the remaining sections of their path. The reduction result for the above example CNF formula is illustrated in Fig. 3. It can be seen that the reduction can be done in polynomial time.

Next, we show that if a 3-CNF formula has a satisfiable assignment, then the constructed TAMP problem has a maximum link load ratio of no more than $(2 + \epsilon)/10$, i.e., $maxRatio \le (2 + \epsilon)/10$. Given a satisfiable assignment, if a variable x (or $\neg x$) is assigned the true value, we let the corresponding flow f_x (or $f_{\neg x}$) place its middlebox on $start_x$, and the negation flow $f_{\neg x}$ (or f_x) on end_x. Note that a 3-CNF formula has a satisfiable assignment if and only if each clause C has at least one variable x assigned the true value, whose corresponding flow f_x will put its middlebox on node $start_x$. Therefore, when f_x arrives at the shared link l_C, its traffic rate is ϵ, and thus the load of l_C is at most $2 + \epsilon$. Since each shared link has a load of no more than $2 + \epsilon$, and any other link has a load of no more than 1, the maximum link load ratio of the entire network is no more than $(2 + \epsilon)/10$.

On the other hand, if the constructed TAMP problem instance has a maximum link load ratio of no more than $(2 + \epsilon)/10$, it indicates that each shared link l_C has at least a flow f_x with its traffic rate being less than one, which means that its middlebox is placed at $start_x$. For each such flow f_x, by assigning the corresponding boolean value x a true value, we obtain a satisfying assignment for the 3-CNF formula. ■

The hardness of TAMP for multiple flows with predetermined paths lies in the factorial number of possible sequences to process the multiple flows. Since different flows are competing for middlebox spaces, flows processed earlier may consume spaces and make them unavailable for flows processed later. We did not find an efficient way to simultaneously process multiple flows except for very limited scenarios with special topologies and traffic changing factors. Alternatively, we will present below a practical heuristic.

The basic idea of the heuristic is to place middleboxes for multiple flows by first processing each individual flow using the LFGL rule and then optimizing middlebox placement between flow pairs. The optimization of multiple flow middlebox placement extends the idea from a single flow to multiple flows. When multiple flows share a common sub-path, we apply the LFGL rule to the middleboxes of those flows on the common path, by placing the middlebox that decreases the maximum amount of traffic at the head of the sub-path, and the middlebox that increases the maximum amount of traffic at the end. However, one difference with LFGL for a single flow is that, since different flows may have different initial traffic rates, we need to consider the amount of traffic change caused by each middlebox, i.e., the product of the traffic rate before entering the middlebox and its traffic changing factor, instead of just the traffic changing factor as in the case for a single flow. Fortunately, by Theorem 1, the middleboxes of a single flow should still be placed in the increasing order of their traffic changing factors, so we know the traffic rate of a flow before entering a middlebox, and thus can obtain the traffic change amount by multiplying its traffic changing factor. For example, if a middlebox of flow f has the i-th least traffic changing factor (ties broken arbitrarily), i.e., $M_f[i]$ in the sorted list, then the flow rate before entering the middlebox is $t_f \prod_{x=1}^{i-1}(1 + alter_{M_f[x]})$, and the its traffic change amount is $\delta_{M_f[i]} = alter_{M_f[i]} t_f \prod_{x=1}^{i-1}(1 + alter_{M_f[x]})$.

The pseudo code to place middleboxes for multiple flows with predetermined paths is shown in Algorithm 1. Brief explanation is as follows. Line 1 sorts all the flows in the decreasing order of their initial traffic rates, inspired by the First-Fit Decreasing Bin Packing algorithm [18] to give priority to large flows. Lines 2 to 5 apply LFGL to each flow, and calculate the traffic change amount of each middlebox of the flow. Line 6 prepares a pair of flows for optimization. Line 7 finds the common sub-paths of the two flows, which may be multiple. Line 8 processes one of such common sub-paths, and line 9 extracts all the middleboxes of the two flows on the common sub-path. Line 10 sorts the extracted middleboxes based on their traffic changing amounts and insert them back to the nodes on the sub-path.

Algorithm 1 Middlebox Placement for Multiple Flows With Predetermined Paths

```
Input: $G, F, route$
Output: $place$
1:  sort $f \in F$ in decreasing order of initial traffic rate $t_f$
2:  for each flow $f$ in $F$ do
3:      apply LFGL to place middleboxes $m \in M_f$ for $f$
4:      calculate traffic change amount $\delta_m$ for each $m \in M_f$
5:  end for
6:  for each pair of flows $f$ and $f'$ do
7:      $P$ = set of common sub-paths between $f$ and $f'$
8:      for each common sub-path $u \sim > v \in P$ do
9:          $M[1..n]$ = extract all middleboxes of $f$ and $f'$ on common sub-path $u \sim > v$
10:         sort $m \in M[1..n]$ in increasing order of traffic change amount $\delta_m$ and insert back in order
11:     end for
12: end for
```

V. Traffic Aware Middlebox Placement Without Predetermined Paths

In this section, we study the general TAMP problem where the flow paths *route* are not predetermined. We first show that the general TAMP problem is NP-hard even for a single flow by reduction from the Hamiltonian cycle problem, and then propose an efficient heuristic by integrating LFGL and MinMax routing that minimizes the maximum link load on the flow path. We also discuss the processing of multiple flows without predetermined paths.

A. NP-Hardness Proof

When flow paths are not determined, the TAMP problem is NP-hard, even for a single flow.

Theorem 3: The general Traffic Aware Middlebox Placement problem is NP-hard.

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

Given an instance of the Hamiltonian Cycle problem with a graph G , we construct an instance of the TAMP problem with a graph $G' = (V', E')$ as follows.

1) For each node $v \in V$, create two nodes $v^{in}, v^{out} \in V'$, where v^{in} has a space capacity of zero, i.e., $sc_{v^{in}} = 0$, and v^{out} of one, i.e., $sc_{v^{out}} = 1$. Connect the two nodes with an edge $(v^{in}, v^{out}) \in E'$, and set its bandwidth capacity to ten, i.e., $bc_{v^{in},v^{out}} = 10$.
2) For each edge $(u, v) \in E$, create an edge $(u^{out}, v^{in}) \in E'$, and set its bandwidth capacity to ten, i.e., $bc_{u^{out},v^{in}} = 10$. An example to create G' from G is shown in Fig. 4.
3) Create a flow f, which is the only flow in F, i.e., $F = \{f\}$. The flow source and destination are both s^{in}, i.e., $s_f = d_f = s^{in}$, where s is an arbitrary node in V. The initial traffic rate is one, i.e., $t_f = 1$. The number of required middleboxes of f is the same as the number of nodes in V, i.e., $|M_f| = |V|$, and each middlebox does not change the volume of processed traffic, i.e., $\forall m \in M_f$, $alter_m = 0$.

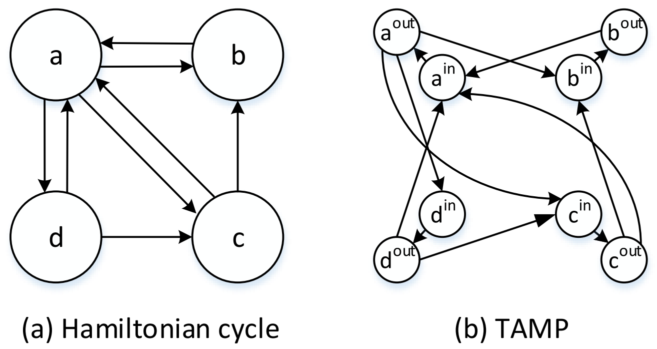


Fig. 4. Reduction from Hamiltonian cycle to TAMP.

Clearly, the above reduction process can be done in polynomial time.

Next, we show that if G has a Hamiltonian cycle, then the TAMP instance with G' and F has a maximum link load ratio of no more than 0.1, i.e., $maxRatio \leq 0.1$. Given the Hamiltonian cycle of G we construct a similar path $route_f$ in G' as follows. Assuming that the Hamiltonian cycle in G starts with s, for each node v and edge (u, v) in the Hamiltonian cycle, add edge (v^{in}, v^{out}) and (u^{out}, v^{in}) to $route_f$, respectively.

Since the Hamiltonian cycle traverses each node in G exactly once, and each node v in G maps to a pair of nodes v^{in} and v^{out} in G', $route_f$ traverses each node in G' exactly once as well. Thus, we can see that $route_f$ has $|V|$ available spaces on the path, sufficient to host all the required middleboxes in M_f. Further, $route_f$ traverses any link in G' at most once, resulting in a maximum link load ratio of 0.1, given that the traffic rate of f is always one.

Reversely, if the TAMP instance with G' and F has a solution $route_f$ and $place_f$ with a maximum link load ratio of 0.1, G will have a Hamiltonian cycle. Given $route_f$ in G', we construct a Hamiltonian cycle in G as follows. Starting with $s_f = s^{in}$, sequentially add the corresponding node $v \in V$ of each incoming node $v^{in} \in V'$ on $route_f$ to the cycle in G. Since $route_f$ traverses all outgoing nodes v^{out} to obtain sufficient middlebox spaces, the constructed cycle traverses all the nodes in G. Further, since the maximum link load ratio in G' is 0.1, $route_f$ traverses each link including (v^{in}, v^{out}) for any v at most once. Thus, the constructed cycle in G traverses each node exactly once, and is a Hamiltonian cycle. ■

Corollary 1: There is no polynomial-time approximation algorithm with an approximation ratio less than two for the general Traffic Aware Middlebox Placement problem unless P=NP.

Proof: By contradiction, assume that there exists a polynomial-time approximation algorithm that achieves an approximation ratio of $C < 2$.

Consider an instance of the Hamiltonian Cycle problem with a graph $G = (V, E)$. Construct an instance of the TAMP problem with a graph $G' = (V', E')$ and a flow set F as in the proof of Theorem 3.

If G has a Hamiltonian cycle, then the constructed TAMP problem has an optimal solution with the maximum link load ratio of 0.1. Thus, the approximation algorithm should return a solution with the maximum link load ratio less than or equal to $0.1 \times C = C < 0.2$.

Otherwise, if G does not have a Hamiltonian cycle, the constructed TAMP problem either does not have a solution or has a solution with the maximum link load ratio of at least 0.2 due to multiple passes of a link.

To sum up, by simply checking whether the maximum link load ratio returned by the approximation algorithm is less than 0.2, we can determine whether the original Hamiltonian cycle problem has a solution within polynomial time, which is a contradiction to the NP-hardness of the Hamiltonian Cycle problem. ∎

The general TAMP problem actually resembles the NP-hard Traveling Salesman Path (TSP) problem [25], which is a generalized version of the Hamiltonian Cycle problem, because TSP allows the source and destination to be different, instead of being the same node, and further the TSP solution needs to minimize the path cost in addition to traversing each node in the network. However, while the set of nodes to traverse in TSP is known, i.e., all nodes, the set of nodes to traverse in TAMP may be a subset of nodes, and there are a combinatorial number of such subsets in the network. Furthermore, even the subset of nodes is known in TAMP, it is the harder non-metric version of TSP, because computer networks generally do not satisfy the triangle inequality [38]. Due to the hardness of the general TAMP problem, we will focus on design efficient and practical heuristics.

B. LFGL Based MinMax Routing for Single Flow

Next, we propose the LFGL based MinMax routing algorithm to calculate the routing path and middlebox placement for a single flow f. The basic idea is to integrate the LFGL rule with the MinMax routing algorithm that minimizes the maximum link load ratio on the flow path.

Based on the LFGL rule, the algorithm also works in two stages. In the first stage, the algorithm traverses the network from the flow source s_f, and iteratively calculates the MinMax path to each node as in Dijkstra's algorithm. When the MinMax path to a node v is determined, the algorithm will attempt to place shrinking middleboxes on v until there is no more space, as if the node is on the selected final path. In addition, the relaxation process will be applied to update the MinMax paths to the neighbors of v. The search will follow multiple candidate paths, and thus the algorithm will attempt placing the same middlebox at different nodes with different candidate paths. The search along a candidate path will terminate if the last shrinking middlebox has been placed on a node, which we call a termination node. When there is no more candidate path to search, the algorithm switches to the second stage to process expanding middleboxes.

In the second stage, the algorithm traverses the network, in a similar way as in the first stage, but backward from the flow destination d_f, and places expanding middleboxes when the MinMax path to a node is found. Note that if all middleboxes

Algorithm 2 Place Middlebox on Node

```
Input: sorted $M_f[1..n]$, $u$, $start$, $flag$
Output: $place_f$, $index(u)$ or $index'(u)$, $t_f^{u+}$ or $t_f^{u-}$
1: $i = start$
2: if $flag < 0$ then
3:     while $sc_u > 0$ and $i \le n$ and $alter_{M_f[i]} \le 0$ do
4:         $place_f(M_f[i], u) = 1$; $sc_u - -$; $i + +$
5:     end while
6:     $index(u) = i - 1$
7:     $t_f^{u+} = t_f^{pred_f(u)+} \prod_{m \in M_f, place_f(m,u)=1} (1 + alter_m)$
8: else
9:     while $sc_u > 0$ and $i \ge 1$ and $alter_{M_f[i]} > 0$ do
10:        $place_f(M_f[i], u) = 1$; $sc_u - -$; $i - -$
11:    end while
12:    $index'(u) = i + 1$
13:    $t_f^{u-} = t_f^{next_f(u)-} / \prod_{m \in M_f, place_f(m,u)=1} (1 + alter_m)$
14: end if
```

are successfully placed, the departure traffic rate $t_f^{d_f+}$ at the destination d_f will be $t_f \prod_{m \in M_f} (1 + alter_m)$, which will be used as the "initial" traffic rate in the second stage. After all expanding middleboxes have been placed along a candidate path, the second stage continues searching the MinMax paths to the remaining nodes, until it reaches a termination node u of the first stage. This means that a path for flow f has been found with two sections: from s_f to u and from u to d_f. For this reason, we call u a junction node. Similar as the first stage, the second stage stops when there is no more path to search.

After the second stage finishes, the algorithm collects all the junction nodes, each corresponding to a different path. The algorithm compares the maximum link load ratio of each path, and selects the path with the minimum maximum link load ratio.

Algorithm 3 shows the pseudo code of the LFGL based MinMax routing algorithm, and Algorithm 2 shows the pseudo code of the *placeMiddlebox* function. The latter places shrinking ($flag = -1$) or expanding ($flag = 1$) middleboxes on node u from the *start*-th one of the sorted list. For easy description, we use $pred_f(u)$ and $next_f(u)$ to represent the preceding and succeeding node of u on the path of flow f, i.e., $route_f(pred_f(u), u) = 1$ and $route_f(u, next_f(u)) = 1$.

Brief explanation of Algorithm 3 is as follows. Line 1 sorts the middleboxes of flow f in the increasing order of their traffic changing factors. Lines 2 to 9 initialize the first stage, where *Saw* is the set of nodes whose MinMax paths from the source s_f have been determined. Line 2 also places shrinking middleboxes on the source s_f until there is no more space or no more shrinking middleboxes. In lines 3 to 9, for each neighbor u of the source s_f, if the link (s_f, u) has more available bandwidth than $t_f^{s_f+}$, then there is a candidate path from s_f to u. $mllr(u)$ records the maximum link load ratio of the candidate path till u. Lines 10 to 18 are the loop to find the MinMax path to a node at a time. At the beginning of each loop, the node $u \notin Saw$ with the minimum maximum link load ratio $mllr(u)$ will be selected and added to *Saw*. Line 12 places shrinking middleboxes on u. Lines 13 to 17 apply the relaxation process,

Algorithm 3 LFGL Based MinMax Routing

```
Input: $G, f, M_f[1..n]$
Output: $route_f, place_f$
1: sort $m \in M_f[1..n]$ in increasing order of $alter_m$
2: Stage 1: $Saw = \{s_f\}$; $placeMiddlebox(s_f, M_f[1..n], 1, -1)$
3: for each neighbor $u$ of $s_f$ do
4:     if $bc_{s_f,u} - l_{s_f,u} \geq t_f^{s_f+}$ then
5:         $mllr(u) = (l_{s_f,u} + t_f)/bc_{s_f,u}$; $pred_f(u) = s_f$
6:     else
7:         $mllr(u) = \infty$
8:     end if
9: end for
10: while $\exists u \notin Saw, index(pred_f(u)) < n$ and $alter_{M_f[index(pred_f(u))+1]} \leq 0$ do
11:     select such $u$ with min $mllr(u)$; $Saw = Saw \cup \{u\}$
12:     $placeMiddlebox(u, M_f[1..n], index(pred_f(u)) + 1, -1)$
13:     for each neighbor $v$ of $u$ do
14:         if $bc_{u,v} - l_{u,v} \geq t_f^{u+}$ and $mllr(v) > \max(mllr(u), (l_{u,v} + t_f^{u+})/bc_{u,v})$ then
15:             $mllr(v) = \max(mllr(u), (l_{u,v}+t_f^{u+})/bc_{u,v})$; $pred_f(v) = u$
16:         end if
17:     end for
18: end while
19: Stage 2: $Saw' = \{d_f\}$; $placeMiddlebox(d_f, M_f[1..n], n, 1)$; $J = \emptyset$
20: if $d_f \in Saw$ and $index(d_f) + 1 = index'(d_f)$ then
21:     $J = J \cup \{d_f\}$; $mllr(d_f) = \max(mllr(d_f), mllr'(d_f))$
22: end if
23: for each neighbor $u$ of $d_f$ do
24:     if $bc(u, d_f) \geq t_f^{d_f-}$ then
25:         $mllr'(u) = (l_{u,d_f} + t_f^{d_f-})/bc_{u,d_f}$; $next_f(u) = d_f$
26:     else
27:         $mllr'(u) = \infty$
28:     end if
29: end for
30: while $\exists u \notin Saw'$ do
31:     select such $u$ with min $mllr'(u)$; $Saw' = Saw' \cup \{u\}$
32:     $placeMiddlebox(u, M_f[1..n], index'(next_f(u)) - 1, 1)$
33:     if $u \in Saw$ and $index(u) + 1 = index'(u)$ then
34:         $J = J \cup \{u\}$; $mllr(u) = \max(mllr(u), mllr(u'))$
35:         continue
36:     end if
37:     for each neighbor $v$ of $u$ do
38:         if $bc_{v,u} - l_{v,u} \geq t_f^{u-}$ and $mllr'(v) > \max(mllr'(u), (l_{v,u} + t_f^{u-})/bc_{v,u})$ then
39:             $mllr'(v) = \max(mllr'(u), (l_{v,u} + t_f^{u-})/bc_{v,u})$;
                $next_f(v) = u$
40:         end if
41:     end for
42: end while
43: if $J \neq \emptyset$ then
44:     select $u \in J$ with min $mllr(u)$; exit with success
45: else
46:     exit with failure
47: end if
```

i.e., checking each neighbor v of u to see whether there is a new path to v via u with a lower maximum link load ratio, and update if yes. Lines 19 to 42 run the second stage in a similar manner, but starting from the flow destination d_f and placing expanding middleboxes. In lines 33 to 36, if the search reaches a node u till which all the shrinking and expanding middleboxes have been placed in the first and second stage, respectively, u will be added to the junction node set J. When the second stage finishes, lines 43 to 47 select from J the node u with the minimum maximum link load, and the final MinMax path can be constructed by tracing $pred_f(u)$ from u to s_f and $next_f(u)$ to d_f. Otherwise, if J is empty, the algorithm fails to find a path.

Since the LFGL based MinMax routing algorithm integrates the LFGL rule and MinMax routing, which is similar to Dijkstra's algorithm, its complexity is the product of the two, i.e., $O((|V| \log |V| + |E|) \times |M_f|)$.

C. *Optimization of Multiple Flows*

Since the general TAMP problem is NP-hard even for a single flow, the problem becomes more challenging for multiple flows. We propose a solution similar to that in Section IV-B, i.e., processing individual flows followed by optimizing middlebox placement between flow pairs. In detail, we first sort all the flows in the decreasing order of their initial traffic rates, to give priority to large flows. Then, we apply the LFGL based MinMax routing algorithm to calculate the routing path and middlebox locations for each individual flow. Finally, we check each pair of flows, identify their common sub-paths, and optimize by swapping middleboxes.

VI. Implementation

Due to its centralized control logic, the emerging Software Defined Networking (SDN) architecture is an ideal platform to implement the proposed algorithms. We have implemented a prototype system using the open-source SDN controller Floodlight [3] and network emulator Mininet [41] to demonstrate our design. In this section, we describe the prototype implementation and discuss real deployment issues.

As explained in Section I, an SDN network is decoupled into the control plane and data plane. For the control plane, we use the modular Floodlight controller, and implement the proposed algorithms as a new module to calculate flow path and middlebox placement. For the data plane, we pick the Mininet network emulator. Mininet can conveniently create a network testbed of hosts and SDN-enabled switches, each as a virtual machine (VM). For each switch in our prototype, we also create an attached NFV server, and connect it with the switch via a high speed link.

Fig. 5 summarizes the workflow of our prototype to process a new incoming flow, and each step is explained in detail below.

A. *Flow Arrival Notification*

Elephant flows can be statically determined based on the application types, such as data backup or VM migration, or dynamically detected using existing techniques in [20] and [42]. When the ingress switch detects or learns an elephant flow, it tries to find a matching entry for the flow in its flow table. If there is no matching entry, the switch wraps a packet of the flow within an *OFPT_PACKET_IN* message, and sends it to the controller. Upon receiving the forwarded packet, the controller learns the arrival of the new flow,

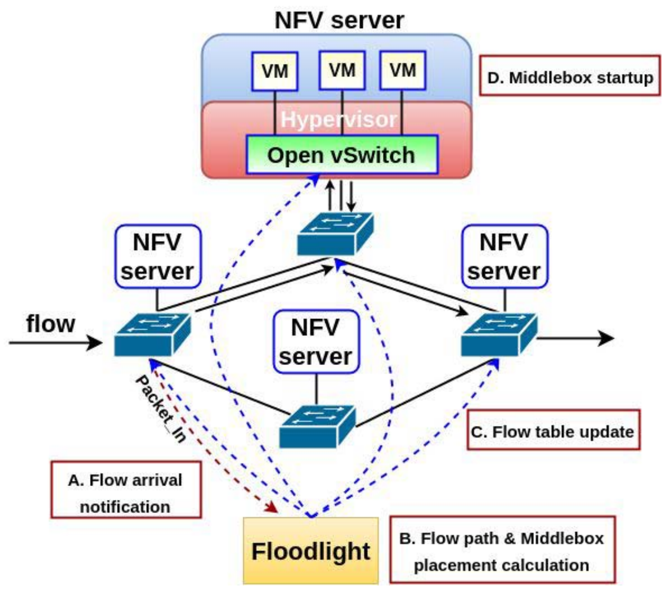


Fig. 5. Middlebox placement workflow.

and will try to calculate a path where each link has a sufficient bandwidth capacity for the flow.

B. Flow Path and Middlebox Placement Calculation

Our module to calculate the flow path and middlebox placement is triggered by the *OFPT_PACKET_IN* message received by the controller. The module determines the application type based on the packet header information, such as IP addresses and port numbers, and determines the set of required middleboxes for the new flow according to predefined profiles. Next, the module applies the proposed algorithms to calculate a flow path and middlebox placement locations.

C. Midllebox VMs Startup

Once the middlebox locations have been calculated, the controller will remotely wake up or start VMs on the selected NFV servers, which can be done through the communication between the VM control software and server hypervisor. For example, for the Kernel-based Virtual Machine (KVM) [4] hypervisor, the command line tool Virsh [10] can be used to remotely start, shutdown, suspend, or resume VMs; for the VMware ESXi hypervisor [11], the VMware vCenter server [12] can be used to remotely control VMs. Predefined VM images for different middleboxes will thus be loaded to perform the desired network functions. Our Mininet prototype focuses on evaluating the network performance, and starts the middlebox VMs in advance.

D. Flow Table Update

After the controller obtains the flow path and middlebox locations, it will accordingly update the switch flow tables to ensure correct packet forwarding. A path is specified in Floodlight as a list of Node-Port tuples in the form of (*DatapathId*, *OFPort*), where *DatapathId* is the Datapath Identity of an OpenFlow instance on a switch and *OFPort* represents a port number of the instance. For the routing path, the controller sends an *OFPT_FLOW_MOD* message to each switch on the routing path. The messages contains the matching fields to define the flow, rule priority, and actions for the flow. In this case, the action is to forward packets of the flow to the calculated output port. On the other hand, for the middleboxes, the controller sends two types of *OFPT_FLOW_MOD* messages to ensure placement locations. The first is to the associated switch, which tells the switch to forward packets to the attached NFV server. The second is to the Open vSwitch (OVS) running in the hypervisor of the NFV server, which instructs the hypervisor to send matching packets to one of the hosted middlebox VMs.

E. Middlebox Emulator Development

To evaluate the effects of middleboxes with different traffic changing factors, we have developed a middlebox emulator program using the *libpcap* library [9]. While a normal TCP/UDP socket will only process packets destined to it, the emulator intercepts all packets forwarded by OVS in the hypervisor using the *libpcap* APIs. To emulate a shrinking middlebox m, the emulator discards intercepted packets with a probability of $-alter_m$. On the other hand, if m is an expanding middlebox, the emulator duplicates intercepted packets with a probability of $alter_m$. In this way, the emulator will accurately change the traffic volume as indicated by the traffic changing factor. After processing, the emulator continues to forward the packets to their actual destinations.

F. Link Load Monitoring

To obtain the link load information necessary for the LFGL based MinMax routing algorithm, we have developed a link load monitoring module in Floodlight. The module periodically sends the *OFPT_STATS_REQUEST* message to every switch to request traffic statistics. Upon receiving the request, a switch will send the *OFPT_STATS_REPLY* response, which contains the transmitted byte count for each port of the switch along with other statistics data. By collecting the information from all switches, the module estimates the load of each link by calculating the exponentially weighted moving average.

G. Post-Processing

After a flow finishes, the flow table entries will be automatically removed after idle or hard timeout [6] by the switch, and the middelbox VMs can be shut down or hibernated after a period of inactivity or manually by the controller.

VII. Experiment and Simulation Results

We use a combination of experiments and simulations to evaluate the proposed algorithms. Experiments in the implemented prototype generate performance data in realistic environments, and simulations in the ns-3 simulator enable us to conduct evaluations in large scale networks with hundreds of hosts. The experiment and simulation programs are available at the project page [7]. In this section, we present extensive experiment and simulation data to demonstrate the effectiveness of our design.

A. Benchmark Solutions

Since there are no existing algorithms for the studied TAMP problem in the literature, we designed the following benchmark solutions. From the proof of Theorem 1, it can be seen that the middleboxes of a flow placed in the increasing order of their traffic changing factors always result in better performance, and thus all the benchmark solutions sort the middleboxes before placement.

In case that the flow paths have been predetermined, there are three benchmark solutions as follows.

- *First-fit:* The first-fit rule starts with the head of the flow path, and places the sorted middleboxes one by one, each at the first available NFV server.
- *Last-fit:* The last-fit rule starts from the end of the flow path, and places the middleboxes sorted in the decreasing order one by one, each at the first available NFV server from the tail, or the last from the head.
- *Random-fit:* The random-fit rule places the middleboxes at random available nodes along the path.

In case that the flow paths are not predetermined, four benchmark solutions are used in the simulations. Each first applies load-balanced (ECMP) shortest path routing to determine the flow path, and then uses LFGL, First-fit, Last-fit, or Random-fit to place middleboxes on the shortest path. Two benchmark solutions are used in the experiments. One is the optimal solution based on the formulation, the other is shortest path routing with LFGL. We did implement First-fit, Last-fit, and Random-fit in this part of the experiments because LFGL has shown superior performance in the previous experiments with predetermined paths.

B. Experimental Results With Prototype

In the prototype experiments, we use the following performance metrics to compare our design and benchmark solutions.

- *End-to-end delay:* The end-to-end delay is the delay between the time points that a packet leaves the source and arrives at the destination. Congestion will result in a longer end-to-end delay.
- *Maximum link load:* The maximum link load is the upper bound of the link load of all the links during the entire experiment run. It is a direct indicator of the congestion level.

For traffic generation, we run Iperf [33] on hosts to generate real constant bit rate (CBR) UDP traffic. We also patched Iperf to be able to measure the end-to-end delay.

1) Effectiveness of LFGL Rule: We first show the effectiveness of the LFGL rule for the TAMP problem with predetermined paths by comparing it with benchmark solutions.

We pick the tree topology, because it is a popular choice for institutional networks, and there is only a predetermined single path between any pair of nodes. We set up a four-layer binary tree with seven switches and eight hosts, as depicted in Fig. 6. Each link has 10 Mbps bandwidth. Each switch u has an attached NFV server with two middlebox spaces, i.e., $sc_u = 2$. We sequentially create four flows from $h1$, $h2$, $h3$,

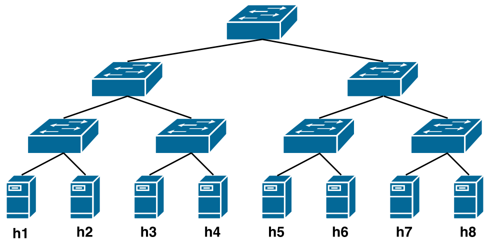


Fig. 6. Tree topology for experiments.

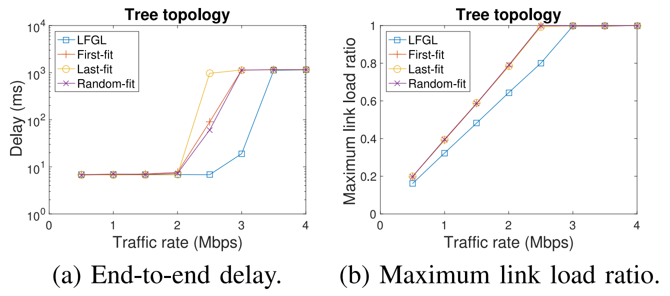


Fig. 7. LFGL experimental results.

and $h4$ to $h5$, $h6$, $h7$, and $h8$, respectively. The initial traffic rate of each flow is adjusted from 0.5 Mbps to 4 Mbps with a stride of 0.5 Mbps. Each flow f requires two middleboxes $M_f = \{m_1, m_2\}$ with traffic changing factors of $alter_{m_1} = -0.2$ and $alter_{m_2} = 0.2$. As a result, the combined traffic changing effect of both middleboxes is $\prod_{m \in M_f}(1 + alter_m) = (1 - 0.2)(1 + 0.2) = 0.96$. When a flow starts, it has only one path, and the locations of its middleboxes will be calculated by the above rules.

Fig. 7(a) shows the average end-to-end delays (in logarithm) of LFGL, First-fit, Last-fit, and Random-fit. We can see that with its optimal placement strategy, LFGL consistently beats the other three rules with shorter delay and postponed congestion. In detail, initially, when the flow rate is small and there is no congestion, all the rules have a small constant delay at about 7 ms. When the flow rate increases to 2.5 Mbps, the delay of LFGL keeps stable, but that of other rules starts increasing due to congestion. This can be explained by the fact that the root is the bottleneck of a tree, and $2.5 = 10/4$ Mbps is the bandwidth available at the root for each of the four generated flows. Specifically, Last-fit has the longest delay at 968 ms, since it puts all middleboxes at the end of flow path, and therefore the flow rate is $1\times$ in most traversed links. On the other hand, First-fit has shorter delay at 89 ms, because it puts middleboxes at the beginning, and thus the flow rate is $0.96\times$ in most traversed links. Using a random strategy, random-fit achieves a short delay of 60 ms. Further, even when the flow rate increases to 3 Mbps, LFGL has a moderate delay of 20 ms. Finally, when the flow rate increases to 3.5 Mbps, all rules have a saturated network with an end-to-end delay of about 1150 ms.

Fig. 7(b) shows the maximum link load ratios of the four rules. Again, LFGL consistently achieves the best performance among the benchmark solutions. For First-fit, Last-fit, and Random fit, their maximum link load ratio is approximately

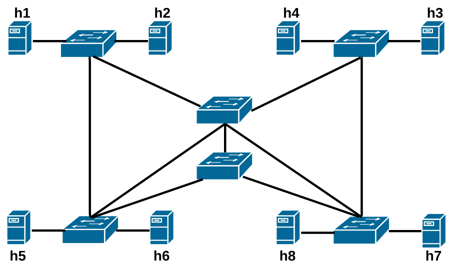
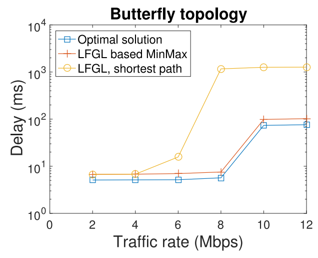
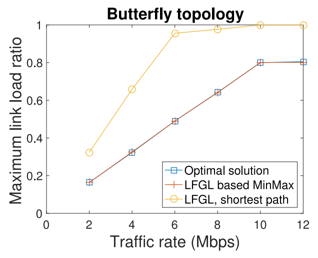


Fig. 8. Multipath topology for experiments.



(a) End-to-end delay.



(b) Maximum link load ratio.

Fig. 9. LFGL based MinMax routing experimental results.

4 times of the ratio between the flow rate and link capacity, and reaches one when the flow rate is 2.5 Mbps. This is consistent with the observation in Fig. 7(a) that congestion happens at 2.5 Mbps for the three rules. On the other hand, the maximum link load ratio of LFGL is approximately 3.2 times of the ratio between the flow rate and link capacity, and reaches one when the flow rate is 3 Mbps. The reason is that LFGL decreases flow rates as early as possible and increase as late as possible, and thus minimizes the traffic volumes in the network core.

2) Effectiveness of LFGL Based MinMax Routing: Next, we demonstrate the effectiveness of the LFGL based MinMax routing algorithm.

We pick the butterfly topology with multiple available paths as depicted in Fig. 8. The network contains six switches and eight hosts. Each switch has two middlebox spaces, and each link has 10 Mbps bandwidth. Four flows are sequentially started from h_1, h_2, h_3, and h_4 to h_7, h_8, h_5, and h_6, respectively. Each flow needs two middleboxes with traffic changing factors of -0.2 and $+0.2$, respectively. We adjust the flow traffic rate from 2 Mbps to 12 Mbps with a stride of 2 Mbps.

Fig. 9(a) shows the average end-to-end delay. We can see that LFGL based MinMax routing outperforms LFGL with shortest path routing due to its traffic awareness in path selection, and achieves performance close to that of the optimal solution calculated from the formulation in Section III. While LFGL based MinMax routing finds disjoint flows to minimize the maximum link load, LFGL with shortest path routing chooses the same shortest path for multiple flows, resulting in earlier congestions and longer delays. The optimal solution produces the shortest delay because it selects the path with the smallest maximum link load ratio as well as least number of hops. In detail, the delay of LFGL with shortest path routing is initially stable at about 7 ms, increases to 16 ms when the flow rate is 6 Mbps, and exceeds 1 second once the flow rate becomes 8 Mbps. On the other hand, the delays of LFGL based MinMax routing and the optimal solution are stable until the flow rate increases to 10 Mbps, and grow to about 102 ms and 96 ms, respectively, when the flow rate reaches 12 Mbps. Fig. 9(b) shows the maximum link load ratio. Although the optimal solution has shorter delays than LFGL based MinMax routing due to its shorter paths, the maximum link load ratio performance of the latter is on a par with that of the former thanks to traffic awareness in path selection. On the other hand, LFGL with shortest path routing saturates much earlier because of overlapping flow paths.

C. Simulation Results With ns-3

In this subsection, we present simulation results in ns-3 to evaluate the proposed algorithms in large scale networks. To better reflect realistic traffic characteristics, instead of using CBR traffic as generated by Iperf, we use the ns-3 On-Off burst traffic model. A flow is in the Off state with no traffic for 50% of the time, and in the On state with continuous CBR traffic for the remaining 50% of the time. The traffic rate of a flow in the On state is the product of a baseline traffic rate and a random number between 0.5 to 1.5. Each flow f requires three middleboxes $M_f = \{m_1, m_2, m_3\}$ with the traffic changing factors of $alter_{m_1} = -0.5$, $alter_{m_2} = -0.2$, and $alter_{m_3} = 0.2$. Each link in the network has a bandwidth capacity of 100 Mbps and propagation delay of 2 ms. Each simulation run lasts 300 seconds, and the presented data are the average of four simulation runs.

In addition to the end-to-end delay and maximum link load, we also collect the following additional performance data.

- *Packet loss ratio:* The packet loss ratio is the ratio of the number of lost packets to the number of sent packets. Heavier congestion will result in a larger packet loss ratio.

1) Topology With Predetermined Paths: We first conduct simulations for topologies with predetermined paths. As in Section VII-B1, we pick the tree topology, and set up a four-layer quad tree with 21 switches and 64 hosts. Each switch has 10 middlebox spaces. Each host generates a flow to a random destination, and we adjust the average traffic rate of each flow from 1.25 Mbps to 12.5 Mbps with a stride of 1.25 Mbps.

Fig. 10(a) compares the average end-to-end delay of LFGL, First-fit, Last-fit, and Random-fit. We can observe a similar trend as in the prototype experiment data that LFGL consistently achieves the shortest delay due to its optimal middlebox placement. By contrast, Last-fit has the longest delay, because it places middleboxes at the end of flow path, and the flow rate is $1\times$ in most links on the path. First-fit achieves shorter delay, since it places middleboxes at the beginning, and thus the flow rate is $(1-0.5)(1-0.2)(1+0.2) = 0.48\times$ in most links. Random-fit has a delay between that of First-fit and Last-fit with a random strategy.

Fig. 10(b) and (c) show the packet lost ratio and maximum link load ratio, respectively, of the four rules. The conclusion is consistent with that in Fig. 10(a) that, in the order of LFGL, First-fit, Random-fit, and Last-fit, each rule achieves a lower packet loss ratio as well as a lower maximum link load ratio than the subsequent ones.

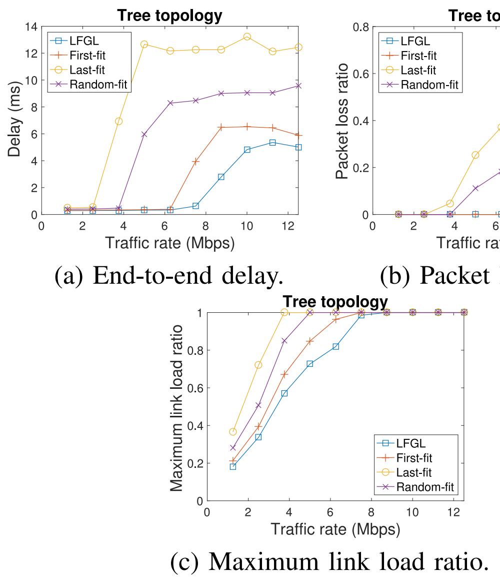


Fig. 10. LFGL simulation results.

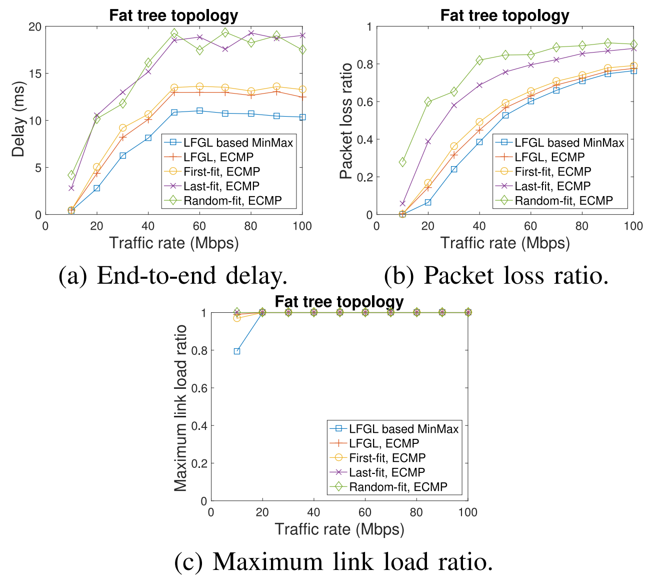


Fig. 12. LFGL based MinMax routing simulation results.

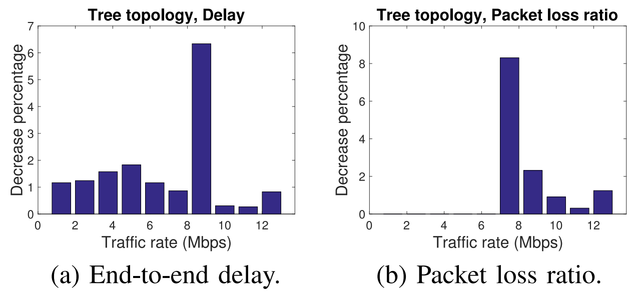


Fig. 11. Improvement of multi-flow optimization with predetermined paths.

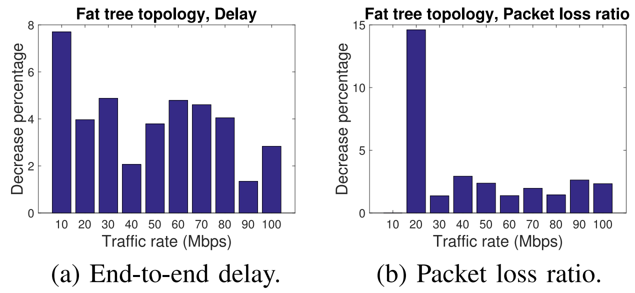


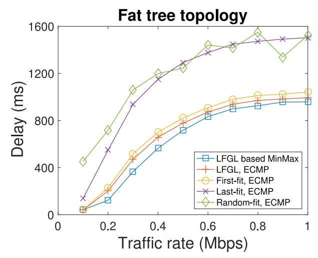
Fig. 13. Improvement of multi-flow optimization without predetermined paths.

We have also evaluated the optimization algorithm for multiple flows presented in Algorithm 1. As comparison, LFGL processes the multiple flows individually in a random order. Fig. 11(a) and (b) show the improvements of the optimization algorithm on the end-to-end delay and packet loss ratio, respectively. We can see that it reduces the end-to-end delay and packet loss ratio by up to 6% and 8%, respectively. The improvement is not significant in some cases, because the optimization algorithm cannot find many middlebox pairs to swap. Note that when the flow rate is less than or equal to 6 Mbps, both solutions have a zero packet loss ratio, and thus there is no improvement. Since the maximum link load ratio measures the instantaneous worst case performance, we do not see significant improvements by the optimization algorithm.

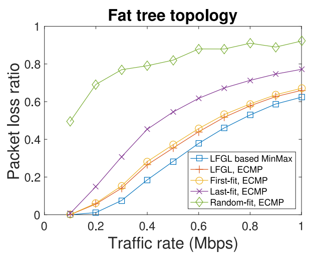
2) Topology Without Predetermined Paths: Next, we consider multi-path topologies, and set up a 8-pod fat tree [13] with 80 switches and 128 hosts. To utilize the multiple paths of the fat tree, we apply equal-cost multi-path (ECMP) [22] load balancing for the shortest-path routing algorithm, by randomly dispatching a flow to one of the available next hops. The traffic rate of each flow is adjusted from 10 Mbps to 100 Mbps with a stride of 10 Mbps. Other settings are similar as in the tree simulations in Section VII-C1.

Fig. 12(a) compares the end-to-end delay of LFGL based MinMax routing with four benchmark solutions. We can see that LFGL based MinMax routing consistently achieves the shortest end-to-end delay. Among the other four benchmark solutions, LFGL achieves a shorter delay than the remaining. Fig. 12(b) shows the packet loss ratio. We can clearly see that in the order of LFGL based MinMax routing, LFGL, First-fit, Last-fit, and Random-fit, each achieves a lower packet loss ratio than the subsequent ones. Finally, Fig. 12(c) illustrates the maximum link load ratio. When the flow rate is 10 Mbps, we see that LFGL based MinMax routing performs the best, but when the flow rate increases to 20 Mbps and above, all algorithms have a saturated instantaneous maximum link load ratio of one.

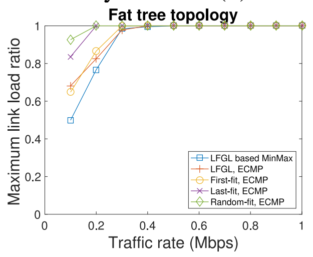
Similarly, we have also conducted simulations to evaluate the optimization algorithm for multiple flows as explained in Section V-C. As comparison, LFGL based MinMax routing processes the multiple flows individually in a random order. Fig. 13(a) and (b) show the improvements of the optimization algorithm on the end-to-end delay and packet loss ratio. We can see that it reduces the end-to-end delay and packet loss ratio by up to 7% and 14%, but is not always effective due to the random order taken by LFGL based MinMax routing to process the flows. Again, when the flow rate is 10 Mbps, both algorithms have a zero packet loss ratio, and thus no improvement is seen in the figure.



(a) End-to-end delay.

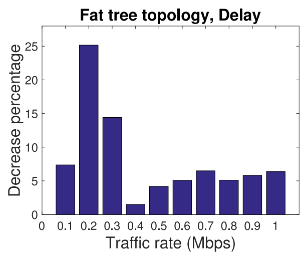


(b) Packet loss ratio.

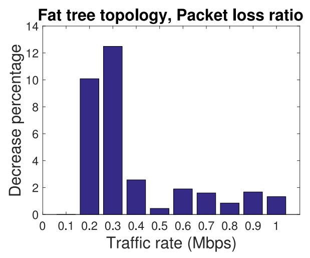


(c) Maximum link load ratio.

Fig. 14. LFGL based MinMax routing simulation results (1024 hosts 1 Mbps link capacity).



(a) End-to-end delay.



(b) Packet loss ratio.

Fig. 15. Improvement of multi-flow optimization without predetermined paths (1024 hosts 1 Mbps link capacity).

For performance evaluation in large scale networks, we have conducted simulations in a fat-tree network with 1024 hosts. To reduce the simulation convergence time, we decrease the link capacity to 1 Mbps. The simulation results for LFGL based MinMax routing are presented in Fig. 14. We can see that the algorithm also works in large scale networks, beating other benchmark solutions. However, the small link capacity leads to a longer transmission delay and consequently a longer end-to-end delay in Fig. 14(a). The data in Fig. 15 also show that our optimization algorithm significantly improves the performance for multiple flows.

D. Comparison Between Experimental and Simulation Results

Comparing the above experimental and simulation results, we can see that they are consistent. In the case with predetermined flow paths, the proposed LFGL rule outperforms other solutions, which rank in the order of First-fit, Random-fit, and Last-fit when the traffic changing ratio product of all middleboxes of a flow is less than one. In the case without predetermined flow paths, LFGL based MinMax routing beats other solutions with shortest path routing.

However, there are also differences due to different link capacities and network sizes. Since the link capacity in the experiments is smaller than that in the simulations, the end-to-end delay in the experiments is long than that in the simulations. Also, the smaller network size and flow number in the experiments result in smoother and more predictable curves due to less variation.

VIII. Conclusion

With the development of virtualization technology, Network Function Virtualization enables flexible deployment of middleboxes as VMs running on commodity server hardware. In this paper, we have studied how to efficiently deploy such middleboxes to achieve load balancing using a Software-Defined Networking approach, and considered in particular the traffic changing effects of different middleboxes. We formulate the Traffic Aware Middlebox Placement (TAMP) problem as a graph based optimization problem, and solve it in two steps. First, we solve the special case of TAMP when flow paths are predetermined. For a single flow, we propose the Least-First-Greatest-Last (LFGL) rule, and prove its optimality; for multiple flows, we prove NP-hardness by reduction from the 3-Satisfiability problem, and propose an efficient heuristic. Next, we solve the general version of TAMP without predetermined flow paths. We prove that the general TAMP problem is NP-hard by reduction from the Hamiltonian problem, and propose the LFGL based MinMax routing algorithm by integrating LFGL with MinMax routing. To validate our design, we have implemented the proposed algorithms in a prototype system with the open-source SDN controller Floodlight and emulation platform Mininet. In addition, we conducted simulations in ns-3 for performance evaluation in large scale networks. Extensive experiment and simulation results are presented to demonstrate the superiority of our algorithms over competing solutions.

References

[1] *Citrix Cloudbridge Technical Overview*. Accessed on Nov. 20, 2016. [Online]. Available: https://www.citrix.com/content/dam/citrix/en_us/documents/products-solutions/cloudbridge-technical-overview.pdf
[2] *Data Center Overlay Technologies*. Accessed on Nov. 20, 2016. [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-730116.html
[3] *Floodlight OpenFlow Controller*. Accessed on Nov. 20, 2016. [Online]. Available: http://www.projectfloodlight.org/floodlight/
[4] *Kernel-Based Virtual Machine*. Accessed on Nov. 20, 2016. [Online]. Available: http://www.linux-kvm.org/
[5] *Network Functions Virtualisation—Introductory White Paper*. Accessed on Nov. 20, 2016. [Online]. Available: https://portal.etsi.org/NFV/NFV_White_Paper.pdf
[6] *OpenFlow Switch Specification Version 1.5.0*. Accessed on Nov. 20, 2016. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf
[7] *Project Homepage*. Accessed on Nov. 20, 2016. [Online]. Available: https://users.cs.fiu.edu/~ pand/TAMP/
[8] *Software-Defined Networking: The New Norm for Networks*. Accessed on Nov. 20, 2016. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf
[9] *TCPDUMP/LIBPCAP Public Repository*. Accessed on Nov. 20, 2016. [Online]. Available: http://www.tcpdump.org/
[10] *Virsh Command Reference*. Accessed on Nov. 20, 2016. [Online]. Available: http://libvirt.org/virshcmdref.html

[11] *VMWare ESXi Hypervisor*. Accessed on Nov. 20, 2016. [Online]. Available: https://www.vmware.com/products/esxi-and-esx/overview

[12] *VMware vCenter Server*. Accessed on Nov. 20, 2016. [Online]. Available: http://www.vmware.com/products/vcenter-server/overview.html

[13] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM*, Seattle, WA, USA, Aug. 2008, pp. 63–74.

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

[15] S. Balon, F. Skivée, and G. Leduc, "How well do traffic engineering objective functions meet TE requirements?" in *Proc. Int. Conf. Res. Netw.*, Coimbra, Portugal, 2006, pp. 75–86.

[16] M. F. Bari *et al.*, "Data center network virtualization: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 2, pp. 909–928, 2nd Quart., 2013.

[17] Z. Bronstein, E. Roch, J. Xia, and A. Molkho, "Uniform handling and abstraction of NFV hardware accelerators," *IEEE Netw.*, vol. 29, no. 3, pp. 22–29, May/Jun. 2015.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.

[19] D. Coudert, A. Kodjo, and T. K. Phan, "Robust energy-aware routing with redundancy elimination," *Comput. Oper. Res.*, vol. 64, pp. 71–85, Dec. 2015.

[20] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *Proc. IEEE INFOCOM*, Shanghai, China, 2011, pp. 1629–1637.

[21] P. Demestichas *et al.*, "5G on the horizon: Key challenges for the radio-access network," *IEEE Veh. Technol. Mag.*, vol. 8, no. 3, pp. 47–53, Sep. 2013.

[22] *IP Multicast Load Splitting—Equal Cost Multipath (ECMP) Using S, G and Next Hop*. Accessed on Nov. 20, 2016. [Online]. Available: http://www.cisco.com/en/US/docs/ios/12_2sr/12_2srb/feature/guide/srbmpath.htmlg

[23] N. Egi *et al.*, "Towards high performance virtual routers on commodity hardware," in *Proc. ACM CoNEXT*, Madrid, Spain, 2008, Art. no. 20.

[24] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul, "FlowTags: Enforcing network-wide policies in the presence of dynamic middlebox actions," in *Proc. ACM HotSDN*, Hong Kong, 2013, pp. 19–24.

[25] U. Feige and M. Singh, "Improved approximation ratios for traveling salesperson tours and paths in directed graphs," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Heidelberg, Germany: Springer, 2007, pp. 104–118.

[26] F. Giroire, F. Havet, and J. Moulierac, "Compressing two-dimensional routing tables with order," *Electron. Notes Discr. Math.*, vol. 52, pp. 351–358, Jun. 2016.

[27] F. Giroire, J. Moulierac, T. K. Phan, and F. Roudaut, "Minimization of network power consumption with redundancy elimination," *Comput. Commun.*, vol. 59, pp. 98–105, Mar. 2015.

[28] E. Gourdin and O. Klopfenstein, "Comparison of different QoS-oriented objectives for multicommodity flow routing optimization," in *Proc. Int. Conf. Telecommun. (ICT)*, 2006, pp. 1–4.

[29] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, Feb. 2015.

[30] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: A GPU-accelerated software router," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 195–206, 2010.

[31] B. Heller *et al.*, "ElasticTree: Saving energy in data center networks," in *Proc. USENIX NSDI*, San Josa, CA, USA, Apr. 2010, p. 17.

[32] J. Hwang, K. K. Ramakrishnan, and T. Wood, "NetVM: High performance and flexible networking using virtualization on commodity platforms," *IEEE Trans. Netw. Service Manag.*, vol. 12, no. 1, pp. 34–47, Mar. 2015.

[33] *IPerf: The TCP/UDP Bandwidth Measurement Tool*. Accessed on Nov. 20, 2016. [Online]. Available: http://sourceforge.net/projects/iperf/

[34] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. New York, NY, USA: Wiley, 1990.

[35] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Rule-caching algorithms for software-defined networks," Dept. Comput. Sci., Princeton Univ., Princeton, NJ, USA, Tech. Rep., 2014.

[36] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, 2000.

[37] G. Lu *et al.*, "ServerSwitch: A programmable and high performance platform for data center networks," in *Proc. USENIX NSDI*, Boston, MA, USA, 2011, pp. 15–28.

[38] C. Lumezanu, R. Baden, N. Spring, and B. Bhattacharjee, "Triangle inequality variations in the Internet," in *Proc. 9th ACM SIGCOMM Conf. Internet Meas. Conf.*, Chicago, IL, USA, 2009, pp. 177–183.

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

[40] 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.

[41] *Mininet: Rapid Prototyping for Software Defined Networks*. Accessed on Nov. 20, 2016. [Online]. Available: http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Mininet

[42] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto, "Identifying elephant flows through periodically sampled packets," in *Proc. ACM IMC*, Taormina, Italy, 2004, pp. 115–120.

[43] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the 'one big switch' abstraction in software-defined networks," in *Proc. ACM CoNEXT*, Santa Barbara, CA, USA, 2013, pp. 13–24.

[44] Z. A. Qazi *et al.*, "SIMPLE-fying middlebox policy enforcement using SDN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 27–38, 2013.

[45] M. Rifai *et al.*, "Too many SDN rules? Compress them with MINNIE," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, San Diego, CA, USA, 2015, pp. 1–7.

[46] 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.

[47] Z. A. Uzmi *et al.*, "SMALTA: Practical and near-optimal FIB aggregation," in *Proc. ACM CoNEXT*, Tokyo, Japan, 2011, Art. no. 9.

[48] T. Wood, K. K. Ramakrishnan, J. Hwang, G. Liu, and W. Zhang, "Toward a software-based network: Integrating software defined networking and network function virtualization," *IEEE Netw.*, vol. 29, no. 3, pp. 36–41, May/Jun. 2015.

[49] X. Zhao, Y. Liu, L. Wang, and B. Zhang, "On the aggregatability of router forwarding tables," in *Proc. IEEE INFOCOM*, San Diego, CA, USA, 2010, pp. 1–9.

Wenrui Ma (S'15) received the B.S. degree in computer science from Zhengzhou University, China, in 2007. She is currently pursuing the Ph.D. degree with the School of Computing and Information Sciences, Florida International University. Her research interests include software-defined networking, network virtualization, and data center networking.

Jonathan Beltran is currently pursuing the bachelor's degree with the School of Computing and Information Sciences, Florida International University. His research interest is in computer networks.

Zhenglin Pan is currently pursuing the bachelor's degree with the School of Computing and Information Sciences, Florida International University. His research interest is in algorithm design.

Deng Pan received the B.S. and M.S. degrees in computer science from Xi'an Jiaotong University, China, in 1999 and 2002, respectively, and the Ph.D. degree in computer science from Stony Brook University in 2007. He is currently an Associate Professor with the School of Computing and Information Sciences, Florida International University. His research interests include high performance network architecture and data center networking.

Niki Pissinou received the B.S. degree in industrial and systems engineering from Ohio State University, the M.Sc. degree in computer science from the University of California, Riverside, and the Ph.D. degree from the University of Southern California. She is currently a Professor with the School of Computing and Information Sciences, Florida International University. Her current research interests include high speed networking, insider attacks detection and prevention in mobile ad-hoc networks, and trust, privacy, and data cleaning mechanisms in trajectory sensor networks.