Traffic-Aware Placement of NFV Middleboxes

Wenrui Ma, Carlos Medina, and Deng Pan
School of Computing and Information Sciences
Florida International University, Miami, FL
wma006@fiu.edu, cmedi045@fiu.edu, pand@fiu.edu

***Abstract*—Network Function Virtualization (NFV) enables flexible deployment of middleboxes as Virtual Machines (VMs) running on general hardware. Different types of middleboxes have the potential to either increase or decrease the volume of processed traffic. In this paper, we investigate the traffic changing effects of middleboxes, and study efficient deployment of NFV middleboxes in Software-Defined Networks (SDNs). To begin with, we formulate the Traffic-Aware Middlebox Placement (TAMP) problem as a graph optimization problem, and show that it is NP-hard when there are multiple flows to consider. Next, by observing that in reality flows arrive one at a time, we leverage the SDN central control mechanism, and propose an optimal solution for the TAMP problem with a single flow. We develop the solution in two steps. First, when the flow path has been determined, we present the Least-First-Greatest-Last (LFGL) rule to place middleboxes. Second, we integrate the LFGL rule with widest-path routing to propose the LFGL based MinMax routing algorithm. Further, we have implemented the proposed algorithm as a module running on top of the Floodlight SDN controller, and conducted experiments in the Mininet emulation system. The experiment results fully demonstrate the superiority of our algorithm over other benchmark solutions.**

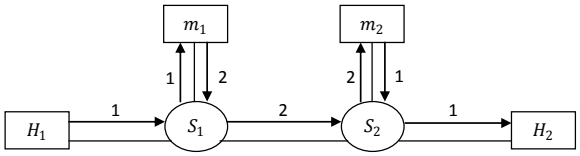
***Index Terms*—Network Function Virtualization, Software-Defined Networking, Traffic-Aware Routing**

I. Introduction

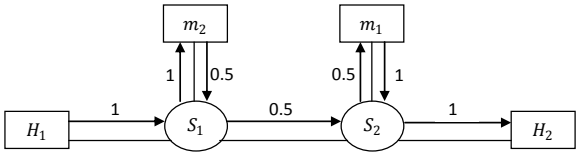
Middleboxes are widely deployed in modern networks. Traditional middleboxes are hardware-based network appliances that implement specialized functions such as firewalls, VPN proxies, and WAN optimizers. Such hardware middleboxes suffer from a number of drawbacks [1], including high cost, function inflexibility, and difficulty to scale up. With the development of virtualization technology, Network Function Virtualization (NFV) emerges as a promising architecture to evolve middlebox implementations by running network function software on virtualized general hardware. The NFV platform is usually an off-the-shelf server that hosts multiple Virtual Machines (VMs), each VM implementing a middlebox function with special software programs. When packets arrive, they are sorted based on predefined traffic processing rules and sent to different software middleboxes.

Different from interconnection network devices such as switches and routers, middleboxes focus on inspecting and manipulating traffic, and thus have the potential to change the volume of processed traffic and may do it in different ways. For example, a WAN optimizer compresses flows before sending them to the next hop. On the other hand, a VPN proxy increases traffic rates due to IPSec header overhead. Finally, a firewall will keep the traffic rates of allowed flows unchanged and reduce the rates of denied flows to zero. For easy description, if a middlebox increases traffic rates, we call it an expanding middlebox, and otherwise a shrinking middlebox.

The following toy example illustrates the traffic changing effects of middleboxes. Consider a network consisting of two switches S_1 and S_2, each with an attached NFV server N_1 and N_2, respectively. Host H_1 and H_2 are connected to S_1 and S_2, respectively, and there is a flow f between them, whose 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. Assume that each of the two NFV servers has space to host only one middlebox. If install m_1 on N_1 and m_2 on N_2, the load of link (S_1, S_2) will be $1 * 2 = 2$, as shown in Fig. 1(a). However, if install m_1 on N_2 and m_1 on N_2, the link load will be $1 * 0.5 = 0.5$, as shown in Fig. 1(b).



(a) m_1 on S_1, m_2 on S_2



(b) m_2 on S_1, m_1 on S_2

Fig. 1: Traffic changing effects of middleboxes.

Previous research on middleboxes has focused on middlebox virtualization on commodity servers [2]–[4], vitualized software middlebox platform [1], placement and chaining of middleboxes in SDN networks [5]–[7]. To the best of our knowledge, the traffic changing effects have not been investigated. In this paper, we study the Traffic-Aware Middlebox Placement problem. We consider an enterprise network with NFV servers to host software middleboxes. Given a set of flows that need be processed by different middleboxes, our objective is to efficiently route the flows and place the middleboxes to achieve optimal network performance.

Our solution leverages the emerging SDN technology [8], and runs as a module on top of the SDN controller. Unlike traditional network architecture, SDN decouples the control

plane and data plane. The control plane makes decisions on where the traffic should be sent, and the data plane forwards the traffic based on the decisions. Consequently, an SDN network consists of a central controller, i.e., the control plane, and a group of switches, i.e., the forwarding plane. By the OpenFlow protocol, the controller and the switches can communicate with each other. When a flow of packets arrive, the controller will set up forwarding rules in switch flow tables, by which the switches will forward the packets to either the next hop or the middleboxes running on attached NFV servers.

Our main contributions in this paper can be summarized as follows. First, we formulate the Traffic-Aware Middlebox Placement (TAMP) problem, and show that it is NP-hard when there are multiple flows. Second, we consider the practical scenario where flows arrive one at a time, and develop an optimal solution in two steps. In the first step, we assume that the flow path has been determined, and propose the Least-First-Greatest-Last (LFGL) rule, which sorts all middleboxes based on their traffic changing ratios, and places shrinking middleboxes from the head of the flow path and expanding middleboxes from the tail. In the second step, we propose the LFGL based MinMax routing algorithm, which integrates the LFGL rule with the widest-path routing algorithm, to find the optimal path that minimizes the maximum link load on the flow path. Finally, we implement the proposed algorithm in the open-source Floodlight SDN controller, and build a prototype with the Mininet emulation system. Experiment results under different topologies are presented to demonstrate the superiority of our algorithm over other benchmark solutions.

The remainder of the paper is organized as follows. Section II formulates the problem. Section III develops the proposed algorithm in two steps. Section IV presents the experiment results. Finally, Section V concludes the paper.

II. Problem Formulation

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

Consider a network represented by a directed graph $G = (H \cup V, E)$, where H, V, and E are the set of hosts, switches, and links, respectively. A switch $u \in V$ may have an attached NFV server, and its space capacity is denoted as $sc(u) \geq 0$, i.e., the number of middleboxes that the NFV sever can host. A link $(u, v) \in E$ has a bandwidth capacity $bc(u, v) \geq 0$, i.e., the available bandwidth.

Let F denote the collection of flows. A flow $f \in F$ is represented as a 4-tuple $(s_f, d_f, t_f^{s_f}, M_f)$ in which $s_f \in H$ is the source host, $d_f \in H$ is the destination host, $t_f^{s_f}$ is the initial traffic rate when f leaves s_f, and M_f is the set of middleboxes that need to be installed for f. Use $t_f(u, v)$ to represent the traffic rate of f on link (u, v). Specifically, $t_f(pre(u), u)$ denotes the traffic rate on the preceding link of u, where $u \in V$ is a switch on the path of f and $pre(u)$ is the preceding node on the path; $t_f(u, post(u))$ denotes the traffic rate on the succeeding link, where $post(u)$ is the succeeding node of $u \in V$.

Denote a middlebox by $m \in M_f$, which is defined as a pair (a_m, k_m), where a_m is the associated processing action, and k_m is the traffic changing factor, or in other words $1+k_m$ is the ratio of the traffic rate of f before and after being processed by m. $k_m \geq -1$ and $1 + k_m \geq 0$, since the traffic rate of a flow cannot be negative. For example, if $m \in M_f$ is installed on $u \in V$, then $t_f(u, post(u)) = t_f(pre(u), u) * (1 + k_m)$. Note that expanding and shrinking middleboxes have positive and non-positive traffic changing factors, respectively.

When a flow f enters the network, by the SDN architecture, the controller will select a path for the flow. Use the decision variable α to denote the flow path as follows:

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

To avoid performance degradation for TCP flows, a flow is not allowed to be split among two paths.

In addition, the controller will select switches to install (on their attached NFV servers) the middleboxes $m \in M_f$. Use the decision variable β to denote the middlebox placement as follows:

$$\beta_m^u = \begin{cases} 1, & \text{if middlebox } m \text{ is installed at node } u. \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The optimization objective is to identify a set of α and β so that the maximum link load is minimized in the network, as shown in Equation (3).

$$\textbf{minimize } maxLinkLoad \tag{3}$$

subject to:

$$\forall (u, v) \in E : \sum_{f \in F} t_f(u, v) \leq maxLinkLoad \leq bc(u, v) \tag{4}$$

$$\forall u \in V, \forall (u, v) \in E, \forall f \in F : t_f(u, v) = \sum_{w \in H \cup V} \alpha_f^{u,v} t_f(w, u) \prod_{m \in M_f} (1 + \beta_m^u k_m) \tag{5}$$

$$\forall u \in V : \sum_{f \in F} \sum_{m \in M_f} \beta_m^u \leq sc(u) \tag{6}$$

$$\forall f \in F : \sum_{u \in V} \alpha_f^{s_f, u} - \sum_{u \in V} \alpha_f^{u, s_f} = 1, \tag{7}$$

$$\sum_{u \in V} \alpha_f^{u, d_f} - \sum_{u \in V} \alpha_f^{d_f, u} = 1 \tag{8}$$

$$\forall f \in F, \forall m \in M_f : \sum_{(u,v) \in E} \beta_m^u \alpha_f^{u,v} = 1 \tag{9}$$

Equation (4) states that, for a link (u, v), the aggregate traffic demand $\sum_{f \in F} t_f(u, v)$ of all flows is no greater than the maximum link load, and should not exceed its bandwidth capacity $bc(u, v)$. Equation (5) states that, for a link (u, v) on

the path of flow f, i.e., $\alpha_f^{u,v} = 1$, the traffic generated by f on this link is the product of the traffic on the previous link $t_f(w,u)$ and the traffic changing ratios $1+k_m$ of all the middleboxes m placed at node u, i.e., $\beta_m^u = 1$. This equation also guarantees flow conservation on each switch on the path. Equation (6) states that, for a switch node $u \in V$, the total number of hosted middleboxes $\sum_{f\in F}\sum_{m\in M_f}\beta_m^u$ should not exceed its space capacity $sc(u)$. Equations (7) and (8) state that flow f must start at its source s_f, and end at its destination d_f. Equation (9) states that a middlebox $m \in M_f$ must be installed on one of the switches u on the path of f.

From the above formulation, we can see that the known NP-hard Multi-Commodity Flow Problem [9] is a special case of the TAMP problem without middleboxes. It can be formally shown that the TAMP problem is NP-hard by reduction from the Multi-Commodity Flow Problem. Due to space limitations, the detailed proof is omitted.

III. Practical Traffic-Aware Middlebox Placement

In this section, we propose a polynomial-time optimal solution for the TAMP problem when there is a single flow, i.e., $|F| = 1$. As seen above, when there are multiple flows, the TAMP problem is NP-hard. Fortunately, in reality, flows tend not to arrive at the exactly same time. Even if multiple flows arrive simultaneously in an SDN network, the controller will have to process them one by one. Thus, our solution for a single flow is of practical importance, especially for SDN networks.

The following theorem shows that for the TAMP problem with a single flow, it is sufficient to find a solution that minimizes the maximum link load on the flow path.

Theorem 1. *For the Traffic-Aware Middlebox Placement problem with a single flow, the solution that minimizes maximum link load on the flow path achieves the global optimization objective.*

Proof. For the purpose of contradiction, assume that the solution α, β, i.e., routing and placement, minimizes the maximum link load on the flow path, but there exists another solution α', β' where $\alpha' \neq \alpha$ or $\beta' \neq \beta$, that has higher maximum link load on its flow path, but lower global maximum link load, i.e., the maximum link load of the entire network instead of the flow path. Note that a link should have the same load in the two solutions, unless it is no the flow path.

We analyze the following three possible scenarios.

1) If the link with the global maximum link load of α, β is not on its flow path, and neither is that of α', β', then the two solutions should have the same global maximum link load.
2) If the link with the global maximum link load of α, β is on its flow path, i.e., equal to its flow maximum link load, since α', β' has higher flow maximum link load, it will have higher global maximum link load as well.
3) If the link with the global maximum link load of α', β' is on its flow path, i.e., equal to its flow maximum link load, since α, β has lower flow maximum link load, it should have lower global maximum link load as well.

In either case, the result contradicts the assumption that α', β' achieves lower global maximum link load. □

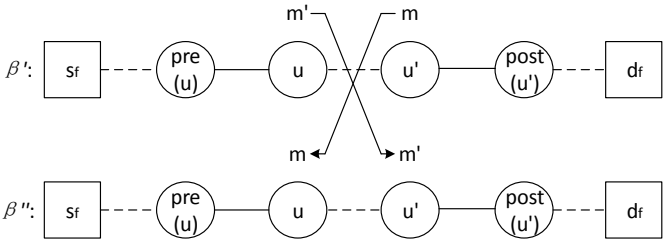


Fig. 2: Proof of Theorem 2.

Next, we develop our solution for a single flow in two steps. First, when the flow path has been determined, we present the Least-First-Greatest-Last (LFGL) rule. It places shrinking and expanding middleboxes from the head and tail of the flow path, respectively, to minimize the amount of traffic entering the network core. Next, we propose the LFGL based MinMax routing algorithm by integrating the LFGL rule with the widest-path routing algorithm. It uses a widest-path like routing algorithm to search the path first from the source and then from the destination, and installs middleboxes simultaneously during the path searching process.

A. Least-First-Greatest-Last Placement Rule

Assume that for a flow f, its path has been determined, i.e., α is known. The only remaining task is then to place middleboxes, i.e., to determine β. The LFGL rule first sorts all the middleboxes $m \in M_f$ based on their traffic changing factors k_m. It then places the middleboxes with non-positive factors or shrinking middleboxes one by one from the head of the path in an increasing order. When one node has used up its space capacity, LFGL continues with the next node on the path. After finishing placing shrinking middleboxes, the rule switches to expanding middleboxes and place them from the path tail in the decreasing order of their traffic changing factors. The deployment succeeds if all middleboxes are placed, and fails otherwise. Due to space limitations, the detailed pseudo code description is omitted.

Theorem 2. *The Least-First-Greatest-Last rule achieves the optimal link load on the flow path.*

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

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 of its placement time, i.e., $\beta_m^u = 1$. By comparison, the other placement scheme installed m on a different node

u', i.e., ${\beta'}_m^{u'} = 1$, which must be after u on the flow path, and instead a different middlebox m' is placed on u, i.e., ${\beta'}_{m'}^{u} = 1$. Apparently, the placement of m' is also different in β and β'. Since among the differences between β and β', m has the least traffic changing factor, we know that $k_m \leq k_{m'}$.

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

$$ {\beta''}_n^v = \begin{cases} {\beta'}_n^v, & \text{if } n \neq m, m' \\ {\beta'}_m^{u'}(=1), & \text{if } n = m, v = u \\ {\beta'}_{m'}^{u}(=1), & \text{if } n = m', v = u' \end{cases} \quad (10) $$

Then, the maximum link load $maxLinkLoad''$ of the new placement β'' will be less than or equal to that of β', i.e., $maxLinkLoad'' \leq maxLinkLoad'$. Denote the traffic rates of f on link $(u, v) \in E$ under β' and β'' as ${t'}_f(u, v)$ and ${t''}_f(u, v)$, 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 β' and β'', the traffic rates of f on such a link are also the same under both schemes, i.e., ${t''}_f(v, w) = {t'}_f(v, w)$.
2) For a link (v, w) between u and $post(u')$, since we exchanged the locations of m and m' in β'', ${t''}_f(v, w) = {t'}_f(v, w)(k_m/k_{m'})$. Since $k_m \leq k_{m'}$ as shown above, we know ${t''}_f(v, w) \leq {t'}_f(v, w)$.
3) For a link (v, w) between $post(u')$ and d_f, since the middleboxes placed after u' are the same under β' and β'', the traffic rates of f on such a link are the same, i.e., ${t''}_f(v, w) = {t'}_f(v, w)$.

Thus, compared with β', β'' achieves no higher maximum link load $maxLinkLoad'' \leq maxLinkLoad'$, and it has one less difference with β generated by LFGL. Continuing this process and eliminating all the difference between β' and β, it can be shown by induction that β achieves no higher maximum link load than that of β', i.e., $maxLinkLoad \leq maxLinkLoad'$, which contradicts the assumption. □

B. LFGL Guided MinMax Routing

Next, we propose the LFGL based MinMax routing algorithm. The basic idea is to integrate the LFGL rule with the widest-path routing algorithm to find the optimal path. Based on the LFGL rule, the algorithm also works in two stages. In the first stage, the algorithm traverses the network starting from the flow source s_f, and calculates the path to each nodes that minimizes the maximum link load. When the MinMax path to a node is determined, the algorithm installs shrinking middleboxes on the node until there is no more space. Note that the calculation process may attempt placing middleboxes at different nodes with a different search path. When all shrinking middleboxes have been placed on each possible searching path, the algorithm switches to process the expanding middleboxes in the next stage.

In the second stage, the algorithm traverses the network backwards from the flow destination d_f, and places expanding middleboxes when the MinMax path to a node is found. Note that if all middleboxes are successfully deployed, the traffic rate at d_f will be $t_f^{s_f} \prod_{m \in M_f} k_m$, which will be used as the "initial" traffic rate when searching path from the path tail. When the second stage reaches a node u that has been visited in the first stage, it means that a path from s_f to d_f has been found where u is the junction node of the two sections of the path. In such a case, the second stage will stop searching along that particular path.

After the second stage finishes, the algorithm collects all the junction nodes and compares the maximum link load of their corresponding paths. The one with the minimum maximum link load will be selected.

The pseudo code description of the LFGL based MinMax routing algorithm is shown in Algorithm 1. For easy description, we define $sc(u)$ to be remaining space capacity of node u, $bc(u, v)$ to be the remaining bandwidth capacity of link (u, v), t_f^u to be the traffic rate of flow f when it leaves node u, i.e., $t_f^u = t_f(u, post(u))$, and ${t'}_f^u$ to be the traffic rate of flow f before it enters node u, i.e., ${t'}_f^u = t_f(pre(u), u)$.

Brief explanation is as follows. Line 1 sorts all the middleboxes in the increasing order of their traffic changing factors. Lines 2 to 9 conduct initialization for the first stage, where Saw is the set of nodes whose MinMax paths from the source have been determined, and $index[u]$ remembers the index of the last middlebox that has been installed on node u. For each neighbor u of the flow source s_f, if the link (s_f, u) has more bandwidth than $t_f^{s_f}$, then there is a possible path from s_f to u. $mll[u]$ records the maximum link load of the path until u, and $pre(u)$ remembers the previous node before u on the path. Lines 10 to 25 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 $mll(u)$ will be selected and added to Saw. Lines 13 to 15 place shrinking middleboxes in order on u until there is no more space or no more shrinking middleboxes. Lines 16 to 18 check whether the newly selected node u is the flow destination d_f and also whether all middleboxes have been placed. If yes, the final MinMax path can be constructed by tracing $pre(u)$ from d_f to s_f, and the algorithm will exit. Otherwise, lines 20 to 24 check each neighbor v of u to see if there is a new path to v via u with lower maximum link load, and update if yes. Lines 26 to 49 run the second stage in a similar manner, but starting from the flow destination and placing expanding middleboxes. In lines 40 to 42, if the second stage reaches a node u that has been visited in the first stage, u will be added to the junction node set J. When both stages finish, lines 50 to 54 select from J the node u with the minimum maximum link load, and the final MinMax path can be constructed by tracing $pre(u)$ to s_f and $post(u)$ to d_f. Otherwise, if J is empty, there does not exist a path that can place all the middleboxes.

Since the LFGL based MinMax routing algorithm integrates the LFGL rule and widest-path routing, its complexity is the product of the two, i.e., $O((|E|+|H \cup V| \log |H \cup V|) \times |M_f|)$.

IV. Experiment Results

We have implemented a prototype using the open-source SDN controller Floodlight [10] and emulation platform

Algorithm 1 LFGL based MinMax Routing

sort $m \in M_f[1..n]$ by k_m in increasing order
Stage one init: $Saw = \{s_f\}; index(s_f) = 0$
for each neighbor u of s_f **do**
  if $bc(s_f, u) \geq t_f^{s_f}$ **then**
    $mll(u) = load(s_f, u) + t_f^{s_f}; pre(u) = s_f$
  else
    $mll(u) = \infty$
  end if
end for
while $\exists u \notin Saw, bc(pre(u), u) \geq t_f^{pre(u)}$ **and** $index(pre(u)) < n$ **and** $k_{M_f[index(pre(u))+1]} \leq 0$ **do**
  select such u with min $mll[u]$; $Saw = Saw \cup \{u\}$
  $i = index(pre(u)) + 1$
  while $sc(u) > 0$ and $i \leq n$ and $k_{M_f[i]} \leq 0$ **do**
    place $M_f[i]$ on u; $sc(u) - -$; $i + +$
  end while
  if $u = d_f$ **and** $i > n$ **then**
    exit with success
  end if
  $index[u] = i - 1; t_f^u = t_f^{pre(u)} \prod_{m \text{ placed on } u} k_m$
  for each neighbor v of u **do**
    if $bc(u, v) \geq t_f^u$ **and** $mll(v) > \max(mll(u), load(u, v) + t_f^u))$ **then**
      $mll(v) = \max(mll(u), load(u, v) + t_f^u); pre(v) = u$
    end if
  end for
end while
Stage two init: $Saw' = \{d_f\}; index'(d_f) = n + 1; t'^{d_f}_f = t_f^{s_f} \prod_{m \in M_f} k_m; J = \emptyset$
for each neighbor u of d_f **do**
  if $bc(u, d_f) \geq t'^{d_f}_f$ **then**
    $mll'[u] = load(u, d_f) + t'^{d_f}_f; post(u) = d_f$
  else
    $mll'[u] = \infty$
  end if
end for
while $\exists u \notin Saw', bc(u, post(u)) \geq t'^{post(u)}_f$ **and** $pre(u) = null$ **do**
  select such u with min $mll'[u]$; $Saw' = Saw' \cup \{u\}$
  $i = index'(post(u)) - 1$
  while $sc(u) > 0$ and $i \geq 1$ and $k_{M_f[i]} > 0$ **do**
    place $M_f[i]$ on u; $sc(u) - -$; $i - -$
  end while
  if $u \in Saw$ **and** $i \leq index(u)$ **then**
    $mll[u] = \max(mll[u], mll[u'])$; $J = J \cup \{u\}$; continue
  end if
  $index'[u] = i + 1; t'^u_f = t'^{post(u)}_f / \prod_{m \text{ placed on } u} k_m$
  for each neighbor v of u **do**
    if $bc(v, u) \geq t'^u_f$ **and** $mll'(v) > \max(mll'(u), load(v, u) + t'^u_f))$ **then**
      $mll'(v) = \max(mll'(u), load(v, u) + t'^u_f; post(v) = u$
    end if
  end for
end while
if $J \neq \emptyset$ **then**
  select $u \in J$ with min $mll[u]$; exit with success
else
  exit with failure
end if

Mininet [11]. In this section, we present the experiment results to demonstrate the effectiveness of our design.

As explained in Section I, an SDN network is decoupled into the control plane and data plane. For the control plane, we use Floodlight as the controller, which is Java-based, modular, and OpenFlow-supported. It can work with both physical and virtual switches. Floodlight has some default modules such as the core module, linkdiscovery module, and topology module that must be loaded each run. For simplicity and efficiency, users can set up Floodlight with minimal module dependencies based on demands. For sepecific network management requirements, new modules can be added to Floodlight to alter the way to control the network. For the data plane, we pick Mininet as a network emulator. Mininet can conveniently create a network testbed, including virtual hosts, Open vSwitches [12], and links. Since each host or switch is a virtual machine, it is possible to run real applications in Mininet to generate traffic. In our implementation, we use Iperf to create pairs of User Datagram Protocol (UDP) data streams from clients to servers and measure the delay and bandwidth of the network.

A. Effectiveness of LFGL Rule

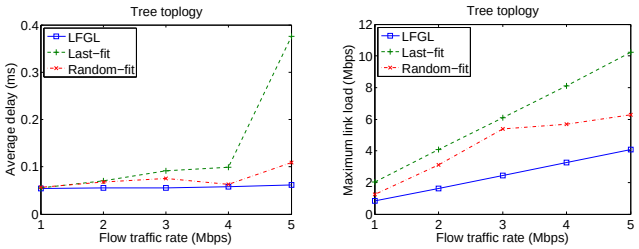
In the first experiment, we compare LFGL with the last-fit and random-fit rules. The last-fit rule sorts middleboxes in the decreasing order of their traffic changing factors, and places them one by one from the tail of the flow path. The random-fit rule randomly places a middlebox at one of the available nodes.

We pick the tree topology in this experiment, since it is a popular topology for enterprise networks, and there is only a single path between any pair of nodes. We use a Python script to create a complete binary tree with depth of three and seven switches in Mininet. Each link has the same bandwidth of 12 Mbps, and each switch can host only two middleboxes. Each leaf-switch has a connected host. We use Iperf to create two UDP flows between two leaf nodes with the greatest distance, and adjust the traffic rate of each flow from 1 to 5 Mbps. Each flow has two associated middleboxes m_1 and m_2 with $k_{m_1} = -0.5$ and $k_{m_2} = -0.2$. The middleboxes are simulated by firewalls that drop the corresponding percentages of traffic.

Figure 3(a) shows the average end-to-end delay of the three benchmark solutions. We can see that LFGL consistently beats last-fit and random-fit. The delay of LFGL is relatively constant, the delay of random-fit increases gradually, while the delay of last-fit jumps dramatically when the traffic rate of each flow increases to 5 Mbps. Figure 3(b) shows the maximum link load of the three solutions. Similarly, LFGL is consistently superior to the other two, and last-fit has the worst performance with the maximum link load equal to the traffic rate sum of both flows.

B. Effectiveness of LFGL based MinMax Routing

In the second experiment, we evaluate our LFGL based MinMax Routing algorithm. The benchmark solutions will first apply the widest-path routing algorithm [13] to find the



(a) End-to-end delay. (b) Maximum link load.

Fig. 3: Comparison under tree topology.

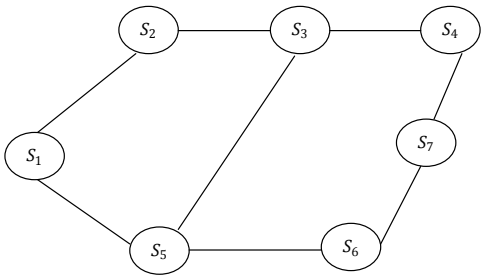
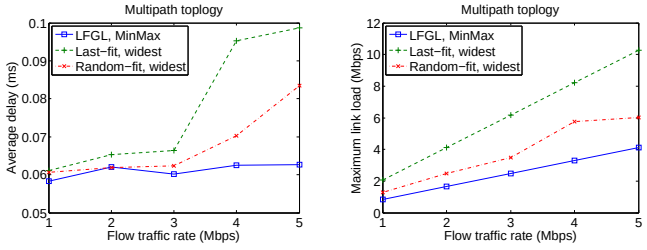


Fig. 4: Multipath topology.

flow path, and then use the last-fit or random-fit rule to place middleboxes.

We use a topology with multiple available paths, as shown in Fig. 4. The network contains seven switches, and each link has 10 Mbps bandwidth. Similar as in the first experiment, we use Iperf to create three UDP flows from S_1, S_2, and S_5 to S_4, respectively. Each flow has two associated middleboxes m_1 and m_2 with $k_{m_1} = -0.5$ and $k_{m_2} = -0.2$, and we adjust the flow traffic rate from 1 to 5 Mbps.

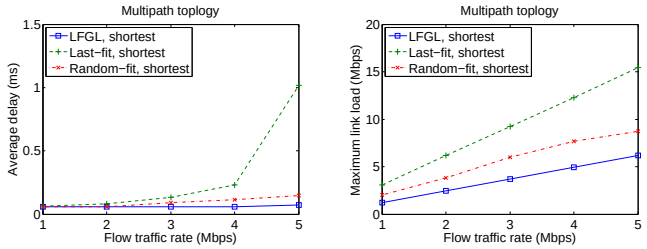
Fig. 5(a) and (b) show the average end-to-end delay and maximum link load of the three competing solutions. We can see that LFGL based MinMax routing consistently beats the other two, while last-fit has the worst performance.



(a) End-to-end delay. (b) Maximum link load.

Fig. 5: Comparison under multipath topology.

In the next experiment, we use similar settings, but calculate the flow path by the shortest-path routing algorithm, and then apply the three rules to place middleboxes. The results are plotted in Fig. 6. By comparing Fig. 5(a) and Fig. 6(a), it can be observed that all the three placement rules achieve much shorter delay with MinMax or widest-path routing. Similarly, comparison of Fig. 5(b) and Fig. 6(b) shows that MinMax and



(a) End-to-end delay. (b) Maximum link load.

Fig. 6: Comparison under multipath topology with shortest-path routing.

widest-path routing achieve lower maximum link load as well.

V. Conclusions

With the development of virtualization technology, NFV enables flexible deployment of middleboxes as VMs running on commodity server hardware. In this paper, we have studied how to efficiently deploy such NFV middleboxes to achieve optimal network performance, and proposed a solution leveraging the emerging SDN technology. First, we formulate the Traffic-Aware Middlebox Placement (TAMP) problem as a graph optimization problem, and show that it is NP-hard in general. Since flows in real networks often arrive one at a time, the solution for the TAMP problem with a single flow is of practical importance. In two steps, we first develop the LFGL rule for the known flow path scenario, and then propose the LFGL based MinMax routing algorithm by integrating LFGL with widest-path routing. To validate our designs, we have implemented the proposed algorithm in the open-source Floodlight SDN controller, and conducted experiments in the Mininet system. The experiment results fully demonstrate the superiority of our algorithm over competing solutions.

References

[1] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco and Felipe Huici, “ClickOS and the Art of Network Function Virtualization,” *NSDI*, 2014.

[2] J.W. Anderson, R. Braud, R. Kapoor, G.Porter, and A. Vahdat, “xOMB: extensible open middleboxes with commodity servers,” *ANCS*, 2012.

[3] V.Sekar, N.Egi, S.Ratnasamy, M.K. Reiter, and G.Shi, “Design and implementation of a consolidated middlebox architecture,” *NSDI*, 2012.

[4] J.Hwang, K. K. Ramakrishnan, and Timothy Wood. ”NetVM: high performance and flexible networking using virtualization on commodity platforms.” NSDI,2014.

[5] Seyed Kaveh Fayazbakhsh et al., “Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags,” *HotSDN*, 2013.

[6] Luizelli, M.C.et al., ”Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions,” IFIP/IEEE International Symposium on Integrated Network Management, 2015

[7] Qazi, Zafar Ayyub, et al. ”SIMPLE-fying middlebox policy enforcement using SDN.” SIGCOMM, 2013.

[8] SDN, https://www.opennetworking.org/sdn-resources/sdn-definition.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 3rd ed. MIT Press, 2009.

[10] Floodlight, http://www.projectfloodlight.org/floodlight/.

[11] Mininet, http://mininet.org/.

[12] Open vSwitch, http://openvswitch.org/.

[13] Deepankar Medhi and Karthikeyan Ramasamy, Network Routing: Algorithms, Protocols, and Architectures, Morgan Kaufmann, 2007.