2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)

Dependable Policy Enforcement in Traditional Non-SDN Networks

Olufemi Odegbile*, Shigang Chen† and Yuanda Wang‡
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, Florida, USA
*Email: *oodegbile, †sgchen, ‡yuandawang@ufl.edu*

***Abstract*—Middleboxes are widely used in modern networks for a variety of network functions in cybersecurity, performance enhancement, and monitoring. Middlebox policy enforcement is however complex and tedious with unreliable manual re-configuration of legacy routers. The existing solution on automated policy enforcement relies on software-defined networking and does not apply to the traditional non-SDN networks, which remain popular today in enterprise deployment and core networks. This paper proposes a new architecture based entirely on *software-defined middleboxes* (instead of using software-defined switches in the prior art) to enable dependable and automated policy enforcement in *non-SDN networks* whose routers forward packets based on traditional routing protocols that are not policy-sensitive. We present a hot-potato enforcement strategy, which is then enhanced with two optimizations for load-balanced policy enforcement. Further enhancements are made to relieve middlebox processing overhead and avoid packet fragmentation due to policy enforcement.**

I. Introduction

Middleboxes offer a powerful and flexible means to augment a network with additional functions (such as firewalling, intrusion detection, proxying, caching and traffic monitoring) which are not provided by the existing routers and other network devices [1], [2]. After middleboxes are deployed into a network, policies are used to define what types of packets should be processed by which middleboxes. Policy enforcement, however, has been a great challenge in traditional networks [3]. Without a central controller, routing paths in traditional networks are determined automatically by distributed execution of routing protocols that are designed primarily for connectivity, stability and efficiency, whereas middlebox policies require selected packets to be forwarded to one or multiple middleboxes and processed in a specific sequence (e.g., FW $\rightarrow$ IDS $\rightarrow$ web proxy). The task of reconciling the difference between policy requirements and underlying routing is complex and tedious, involving unreliable and error-prone manual re-configuration of legacy routers.

One class of solutions relies on the use of software-defined networking (SDN) to provide dependable enforcement of middlebox policies [2], [4], where software-defined switches enable dynamic flow-level routing and a central controller is responsible for establishing the forwarding paths of individual flows through middleboxes based on policy requirements. Given a set of policies, the controller configures the SDN switches with specific forwarding rules and tunnel rules that direct the policy-matching packets to appropriate middleboxes in proper orders for processing. However, these prior solutions work only with SDN networks. They are not intended for non-SDN traditional networks, which remain popular on today's Internet, particularly for enterprise networks and core networks. Without SDN switches, the routers in a traditional network do not support flow-level routing. Our goal is to enable dependable and automated enforcement of middlebox policies on *traditional non-SDN networks* whose routers forward packets based on classical routing protocols such as OSPF [5] and EIGRP [6].

Since we cannot rely on software-defined switches, our idea is to introduce *software-defined middleboxes* (SDM), either implemented by software on general-purpose computers or by hardware on specialized devices, which are deployed as an augmentation to an existing network either *off-path* by connecting to a subnet off a router as an IDS is typically installed or *in-path* by sitting between two routers as a firewall is typically installed. These software-defined middleboxes are configured by a central *middlebox controller* that has the knowledge of network topology, placement of the middleboxes, policies, and traffic measurements reported from the middleboxes. At the architectural level, in the prior SDN solutions [2], [4], their controller configures the SDN switches to enforce the policies; in our SDM solution, the controller cannot configure the routers (which are not SDN-capable) but instead configures the SDMs for policy enforcement over a traditional network where routers and other network devices perform their operations oblivious to policy enforcement. Unlike SDN controllers, our controller is not involved in determining switch-to-switch packet forwarding at individual flow level. It only pre-configures the middleboxes based on user-specified policies. It is not invoked as individual flows enter the network, and thus is unlikely to become a bottleneck.

The contributions of this paper are summarized as follows: First, we introduce a new architecture for dependable and automated middlebox policy enforcement in traditional non-SDN networks. Our solution replaces the need for *software-defined switches* in prior art [4] with the use of *software-defined middleboxes*, making policy enforcement transparent to an existing, traditional network. Second, we present a simple hot-potato enforcement strategy as a basic framework

2575-8411/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDCS.2019.00061

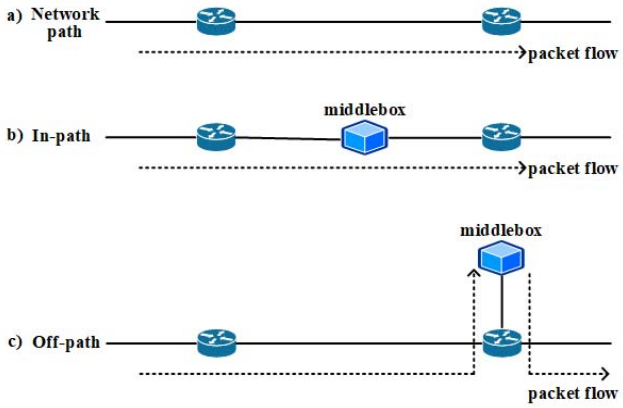


Figure 1: a) a network path; b) insert a middlebox in the path; c) insert a middlebox off the path.

for policy enforcement by software-defined middleboxes. Third, we propose a load-balanced enforcement strategy with two linear program optimizations that help to prevent unbalanced traffic distribution from overloading some middleboxes while under-utilizing others. Fourth, we provide alternative designs that relieve middlebox processing overhead and avoid packet fragmentation due to policy enforcement. Finally, we evaluate the effectiveness of the proposed solution in load-balanced policy enforcement.

The rest of the paper is organized as follows. Section II formulates our research problem. Section III presents the detailed architecture of our solution, a hot potato packet-forwarding strategy, an optimized load balancing strategy, and other enhancement techniques. Section IV evaluates the proposed software-defined middlebox solution. Section V discusses the related work. Section VI concludes the paper.

II. Network Model and Problem Statement

Consider a large non-SDN enterprise network, consisting of (1) edge routers that connect stub networks and (2) core routers that interconnect the edge routers. Assume that the routers run OSPF [5] to establish their routing tables for packet forwarding. We study how to enforce network management policies automatically in a traditional network with the help of software-defined middleboxes. A middlebox [4] is a network device that is deployed in path (Figure 1.b) or off path (Figure 1.c) to implement a network function such as firewalling, intrusion detection, proxying, measurement or access control in software or hardware. A middlebox that is dynamically reconfigurable based on policies is called a *software-defined middlebox*.

A policy consists of a traffic descriptor and an ordered action list. The traffic descriptor contains a number of packet-header fields, and wildcards may be used as masks to allow a policy to match multiple traffic flows. Table II gives six policy examples, where only four header fields and the action list are shown, with other fields being wildcards by default. Let subnet a be the address prefix for the enterprise network, e.g., 128.40.*.*. The first two polices state that the internal web traffic is permitted without further action. The next two policies state that web access from external hosts to internal web servers must go through firewall and intrusion detection. The following two policies state that web access from internal hosts to external web servers must go through firewall, intrusion detection, and web proxy.

Table I

src addr	dst addr	src port	dst port	action list
subnet a	subnet a	*	80	permit
subnet a	subnet a	80	*	permit
*	subnet a	*	80	FW, IDS
subnet a	*	80	*	IDS, FW
subnet a	*	*	80	FW, IDS, proxy
*	subnet a	80	*	proxy, IDS, FW
...	...	...	...	...

Given a network and an ordered list of n networkwide policies $P = \{\langle d_i, a_i\rangle \mid 1 \leq i \leq n\}$ where d_i is a traffic descriptor and a_i is an ordered action list, the *problem of policy enforcement* is that, for each packet that matches a policy $\langle d_j, a_j\rangle$, we make sure that the actions in a_j are performed on the packet in the specified order. When there are multiple policy matches, we apply the first matching policy to the packet.

Like [4], we study how to use middleboxes (which implement the required actions, i.e., network functions) to enforce network management policies. Unlike [4], which is designed for SDN switches, we consider policy enforcement in traditional enterprise networks that are not SDN-capable. Such networks still dominate on today's Internet, and the methods developed in [4] are not applicable to them.

III. Policy Enforcement by Software Defined Middleboxes

In this section, we begin with a new system architecture that includes a controller and policy proxies to support automated policy enforcement by software-defined middleboxes deployed in traditional networks. We then introduce a baseline hot-potato enforcement strategy and improve it through a series of optimizations and augmentations.

A. New System Architecture

As illustrated in Figure 2, the proposed system is composed of a centralized management server called the *controller*, a policy proxy for each stub-network, and software-defined middleboxes. Our controller is different from the controller in SDN networks; it manages the middleboxes, not SDN switches. A stub-network is a subnet that does not

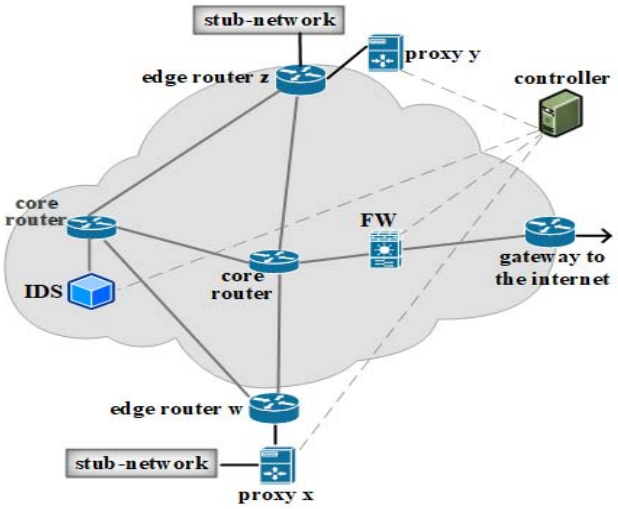


Figure 2: a) a network path; b) insert a middle box in the path; c) insert a middle box off the path.

route transient traffic and is connected to the core network through an edge router. A policy proxy identifies traffic that is subject to policies, and it assists in policy enforcement. Analogous to a web proxy on web traffic, a policy proxy will intercept traffic for policy compliance.

A policy proxy may be considered as a special software-defined middlebox configurable by the controller. It can be connected to the network in one of two ways: off-path or in-path. Edge router z in Figure 2 connects to an off-path proxy y. Router z is configured with a loopback interface that forwards all received packets to proxy y and after receiving these packets back, performs regular routing-table lookup and packet forwarding. Proxy x is connected in-path to edge router w, and in this case the policy enforcement is completely transparent to router w.

Similarly, the middleboxes that implement network functions can be connected to routers in the above two ways. No loopback will be performed by a core router. Hence, in our design, no matter whether a middlebox is in-path (FW in the figure) or off-path (IDS in the figure), they are transparent to the core routers, which are policy-unaware.

Some notations are defined as follows: Let S be the set of stub-networks (or simply referred to as subnets), each behind a policy proxy, M be the set of all middleboxes, and R be the set of all policy proxies. Let Π be the set of network functions that the middleboxes implement. Consider an arbitrary proxy or middlebox, denoted as x. Let Π_x be the set of network functions that x does **not** implement. If x is a proxy, then $\Pi_x = \Pi$. For an arbitrary network function $e \in \Pi$, we use M^e to denote the subset of middleboxes that implement e. The notations defined above and later are summarized in Table II for quick reference.

Table II

Notations	Definition
S	set of stub-networks
M	set of all middleboxes
R	set of all policy proxies
Π	set of network functions the middleboxes implement
P	list of networkwide policies
x	an arbitrary proxy or middlebox
e	an arbitrary network function
Π_x	set of network functions not implemented by x
M^e	set of all middleboxes offering $e \in \Pi$
m_x^e	the closest middlebox to x offering e
M_x^e	a subset of middleboxes, which offers e and are assigned to x
P_x	a subset of policies whose descriptors overlap the subnet address of x
$P_{s,d}$	set of policies that match any source address in s and any destination address in d, $s, d \in R, s \neq d$
T_p	traffic volume of flows matching $p \in P$
$T_{s,p}$	traffic volume from s that matches p
$T_{d,p}$	traffic volume received by d that matches p
$T_{s,d,p}$	traffic volume from s to d that matches p
$C(x)$	processing capacity of a middlebox $x \in H$

B. Hot-Potato Enforcement Strategy

The controller is configured with the complete network topology with subnet addresses, the placement of all middleboxes, and the user-specified policies. Consider an arbitrary middlebox or policy proxy x. For each function $e \in \Pi_x$, the controller finds a middlebox m_x^e that implements e and is closest to x. This can be easily done using a shortest-path algorithm such Dijkstra's algorithm [7]. The controller sends m_x^e, $\forall e \in \Pi_x$, to middlebox/proxy x so that x knows where to forward a packet that requires function x.

In addition, if x is a policy proxy, the controller finds the subset P_x of policies whose traffic descriptors contain at least one source address from the subnet behind x. It informs the proxy x of these relevant policies in P_x. If x is a middlebox, the controller finds the subset P_x of policies whose action lists contain any function that x performs. It informs the middlebox x of its relevant policies as well. P_x will be stored in the local policy table at the recipient proxy or middlebox.

We now describe the enforcement operations: When a proxy x receives an outbound packet, it matches the relevant packet-header fields against the traffic descriptors in its policy table P_x. Policy matching is outside the scope of this paper; there is a large body of literature on how to perform multi-field matching in software or hardware [8], [9], [10]. If the packet does not match any policy in P_x, it is forwarded to the connected edge router. Otherwise, let p be the first policy in P_x that matches the packet. The proxy finds the first network function e in the action list of p. It tunnels

the packet IP-over-IP [7] to the closest middlebox m_x^e that implements the function, traditionally referred to the hot-potato strategy in packet forwarding. More specifically, the proxy adds a new IP header on top of the original one, with its own address as the source and m_x^e as the destination. The packet will be routed to m_x^e, where the outer IP header will be striped and the required action e will be taken. After that, m_x^e will match the packet against its policy table to find a matching policy (which will be p), it identifies the next function e' in the list, and tunnel the packet IP-over-IP to the closest middlebox for function e'. The process is repeated until the last function in the list is performed by a middlebox, which will then simply forward the original packet (without an outer IP header any more) to its connected router. From there, the packet takes its normal routing path towards its destination.

The above approach has three problems. First, forwarding packets to the middleboxes m_x^e on the topologically shortest paths does not take the workload into consideration, which may overload some middleboxes while letting others idle. Second, it is inefficient for each middlebox to perform multi-field matching for every received packet. This may overload the middleboxes in the core and create throughput bottlenecks. Third, IP-over-IP increases packet size and may cause packet fragmentation, which causes additional overhead. In the sequel, we will solve these problems.

C. Load-balanced Enforcement Strategy

To solve the first problem, we allow each middlebox x more flexibility in forwarding packets towards the next-hop middleboxes. Instead of assigning a single middlebox m_x^e for each function $e \in \Pi_x$, we can assign a subset of M^e, denoted as M_x^e, from which x can choose one to forward packets that require the next function e. For example, M_x^e may consist of k members from M^e that are closest to x, where $k(\geq 1)$ is a system parameter that can be arbitrarily set; when $k = 1$, it becomes the hot-potato enforcement strategy.

While all packets that arrive at x and require the network function e can be forwarded to any member in M_x^e, our goal is to distribute such traffic among the members of M_x^e to achieve load-balanced policy enforcement. Periodically, all policy proxies send their measured traffic volumes to the controller. Consider an arbitrary source subnet s and an arbitrary destination subnet $d \neq s$. Let $P_{s,d}$ be the set of policies that match at least one source address in s and at least one destination address in d. Let $T_{s,d,p}$, $\forall s, d \in R, s \neq d, p \in P$, be the traffic volume from s to d that matches p, which can be measured at the policy proxy of source s and reported by the proxy to the controller.

With the above traffic measurements, the controller solves a load-balancing linear programming optimization to assign a traffic volume between any two middleboxes, denoted as $t_{s,d,p}(x, y)$, for the amount of traffic in $T_{s,d,p}$ that should be sent from middlebox x to middlebox y.

Before formulating the load-balancing optimization to determine $t_{s,d,p}(x, y)$, we introduce three binary indicators: $I_{s,d,p}(e, e^{'})$, $J_{s,d,p}(e)$ and $J^{'}_{s,d,p}(e)$. Specifically, $I_{s,d,p}(e, e^{'})$ is set to 1 if (1) p matches any source address from s and any destination address from d and (2) e and $e^{'}$ are two adjacent functions in the action list of p, or 0 otherwise. $J_{s,d,p}(e)$ is set to 1 if (1) p matches any source address from s and any destination address from d and (2) e is the first network function in the action list of p, or 0 otherwise. $J^{'}_{s,d,p}(e)$ is set to 1 if (1) p matches any source address from s and any destination address from d and (2) e is the last network function in the action list of p, or 0 otherwise. Let $C(x)$ be the processing capacity of a middlebox $x \in M$, and λ be the largest load factor among all middleboxes. Our load-balancing optimization problem is

$$
\begin{aligned}
&\min \quad \lambda \\
&\text{s.t.} \\
&\textstyle\sum_x t_{s,d,p}(x,y) = \sum_{z \in M_y^{e'}} I_{s,d,p}(e,e^{'}) \cdot t_{s,d,p}(y,z) + \\
&\qquad \textstyle\sum_{(s,d,p)} J^{'}_{s,d,p}(e) \cdot t_{s,d,p}(y,d), \quad \forall e, e^{'}, s, d, yp \\
&\textstyle\sum_{x \in M^e} \sum_{y \in M_x^{e'}} t_{s,d,p}(x,y) \; = \\
&\qquad I_{s,d,p}(e,e^{'}) \cdot T_{s,d,p}, \quad \forall e, e^{'}, s, d, p \\
&\textstyle\sum_{y \in M_s^e} t_{s,d,p}(s,y) = J_{s,d,p}(e) \cdot T_{s,d,p}, \quad \forall e, s, d, p \\
&\textstyle\sum_{x \in M^e} t_{s,d,p}(x,d) = J^{'}_{s,d,p}(e) \cdot T_{s,d,p}, \quad \forall e, s, d, p \\
&\textstyle\sum_{e' \in \Pi_x} \sum_{(s,d,p), y \in M_x^e} I_{s,d,p}(e,e^{'}) \cdot t_{s,d,p}(x,y) + \\
&\quad \textstyle\sum_{(s,d,p)} J^{'}_{s,d,p}(e) \cdot t_{s,d,p}(x,d) \leq \lambda \cdot C(x), \quad \forall x \in M \\
&t_{s,d,p}(x,d) \; \geq 0, \quad \forall x \in M \\
&t_{s,d,p}(x,y) \; \geq 0, \quad \forall e, e^{'}, x \in M^e, y \in M_x^{e'} \\
&t_{s,d,p}(x,y) \; = 0, \quad \forall e, e^{'}, x \in M^e, y \notin M_x^{e'} \\
&\lambda \; \leq 1,
\end{aligned}
\tag{1}
$$

The first constraint ensures flow conservation at each middlebox y for traffic $T_{s,d,p}$ from source s to destination d that matches policy p. The left side of the constraint is the sum of the traffic matching (s, d, p) that y receives, and the right is what y forwards. The second constraint ensures that the total traffic volume sent from source s to other middleboxes towards destination d matching p is equal to $T_{s,d,p}$. The third constraint ensures that the total traffic volume sent by source s to destination d and matching p is $T_{s,d,p}$. Likewise, the fourth constraint ensures that the total traffic volume received by destination d from source s that matches p is also $T_{s,d,p}$. The fifth constraint makes sure that the capacity of each middlebox is not exceeded, where λ is the largest load factor among all middleboxes. We want to minimize λ. Eq. (1) is a linear programming problem and

can be solved in polynomial time.

After $t_{s,d,p}(x,y)$ for all s,d,p,x,y combinations is determined, the controller sends each middlebox/proxy x the values of $t_{s,d,p}(x,y)$, $\forall s,d,p,y$. After receiving these values, when x receives a packet that matches policy p from a source address in s and a destination address in d, it forwards the packet to a middlebox y selected from M_x^e with a probability proportional to $t_{s,d,p}(x,y)$, where e is the next function in the action list of p; if there is no next function, the packet is forwarded towards the destination address. To implement the probabilistic middlebox selection, x hashes the flow identifier of the packet (including source address, destination address, source port, destination port, and protocol ID from the packet header). Let r be the hash output in the range of $[0,N)$, where N is the maximum hash value. Let $M_x^e = \{y_1, y_2, ..., y_k\}$. Middlebox y_i will be selected if $\sum_{j=1}^{i-1} t_{s,d,p}(x,y_j)/\sum_{j=1}^{k} t_{s,d,p}(x,y_j) \leq r/N < \sum_{j=i}^{i} t_{s,d,p}(x,y_j)/\sum_{j=1}^{k} t_{s,d,p}(x,y_j)$.

The number of decision variables, $t_{s,d,p}(x,y)$ for all s,d,p,x,y combinations in (1), can be numerous in a large network with many policies. Although the calculation is done offline by the controller, we offer a different problem formulation to reduce the number of decision variables and consequently reduce the computation overhead at the controller as well as the communication overhead for the controller to send these values to the middleboxes.

Consider an arbitrary function $e \in \Pi$ and an arbitrary proxy/middlebox x with $e \in \Pi_x$. Let $t_{e,p}(x,y)$, $\forall y \in M_x^e$, be the volume of all traffic from x to y that matches p and requires the next function e. Let $t_p(x,d)$ be the size of all traffic that matches p and is sent directly from middlebox x to a destination d. The controllers solve a simplified load-balancing linear programming optimization to assign (1) $t_{e,p}(x,y)$, an aggregate volume of traffic matching p between any two middleboxes x and y for function e and (2) $t_p(x,d)$, an aggregate volume of traffic matching p between any middleboxe x and a destination d. Similar to the previous formulation, we introduce three binary indicators: $I_p(e,e^{'})$, $J_p(e)$ and $J_p^{'}(e)$, $\forall e,e' \in \Pi, e \neq e', p \in P$. Specifically, $I_p(e,e^{'})$ is set to 1 if e and $e^{'}$ are two adjacent functions in the action list of p, or 0 otherwise. $J_p(e)$ is set to 1 if e is the first network function in the action list of p, or 0 otherwise. $J_p^{'}(e)$ is set to 1 if e is the last network function in the action list of p, or 0 otherwise.

Let T_p be the total traffic volume that matches p regardless of source and destination. Let $T_{s,p}$ be the traffic volume from s that matches p, and $T_{d,p}$ the traffic volume received by d that matches p. They can be measured at the policy proxies and tallied at the controller. The new load-balancing optimization is formulated as follows:

$$
\begin{aligned}
&\min \quad \lambda \\
&\text{s.t.} \\
&\textstyle\sum_x t_{e,p}(x,y) = \sum_{e' \in \Pi_y} \sum_{z \in M_y^{e'}} I_p(e,e^{'}) \cdot t_{e^{'},p}(y,z) \\
&\qquad\qquad + \textstyle\sum_{d \in R} J_p^{'}(e) \cdot t_p(y,d), \quad \forall e,e^{'},p,y \\
&\textstyle\sum_{x \in M^e} \sum_{y \in M_x^{e'}} t_{e^{'},p}(x,y) = I_p(e,e^{'}) \cdot T_p, \quad \forall e,e^{'},p \\
&\textstyle\sum_{x \in M^e} \sum_{d \in R} t_p(x,d) = J_p^{'}(e) \cdot T_p, \quad \forall e,p \\
&\textstyle\sum_{x \in M_s^e} t_{e,p}(s,x) = J_p(e) \cdot T_{s,p}, \quad \forall s,e,p \\
&\textstyle\sum_{x \in M^e} t_p(x,d) = J_p^{'}(e) \cdot T_{d,p}, \quad \forall d,e,p \\
&\textstyle\sum_{e' \in \Pi_x} \sum_{p \in P, y \in M_x^e} I_p(e,e^{'}) \cdot t_{e^{'},p}(x,y) \\
&\qquad + \textstyle\sum_{p \in P, d \in R} J_p^{'}(e) \cdot t(x,d) \leq \lambda \cdot C(x), \quad \forall x \\
&t_p(x,d) \qquad \geq 0, \quad \forall x \in M, d \in R \\
&t_{e^{'},p}(x,y) \qquad \geq 0, \quad \forall e,e^{'}, x \in M^e, y \in M_x^{e^{'}} \\
&t_{e^{'},p}(x,y) \qquad = 0, \quad \forall e,e^{'}, x \in M^e, y \notin M_x^{e^{'}} \\
&\lambda \ \leq 1
\end{aligned}
\tag{2}
$$

Similar to the previous formulation, the first constraint ensures flow conservation at each middlebox. The second and third constraints ensure flow conservation for each policy p after implementing each networking function in it action list. The fourth constraint ensures that the total volume of traffic matching p sent from source s to all middleboxes offering function e is equal to the total measured volume. Likewise, the fifth constraint ensures that the total volume of traffic matching p received by destination d from all middleboxes offering function e is equal to the total measured volume. The sixth constraint makes sure that the capacity of each middlebox is not exceeded. Eq. (2) is again a linear programming problem and can be solved in polynomial time.

After $t_{e,p}(x,y)$ for all e,p,x,y combinations is determined, the controller sends each middlebox/proxy x the values of $t_{e,p}(x,y)$ for different functions e, different policies p, and different next-hop middleboxes y. After receiving these values, x forwards all packet matching a policy p with the next function being e to a middlebox y selected from M_x^e with a probability proportional to $t_{e,p}(x,y)$.

D. Avoid Multi-field Policy Matching at Middleboxes

Policies may contain wildcards as the examples in Table II show. TCAM can be used to support lookups of such policies. However, the size of TCAM is typically small. In order to support a large number of policies, we may resort to software lookups using trie-based data structures [11] to implement the policy table, which is slower than TCAM and may cause throughput bottleneck in the core network under heavily traffic load conditions.

We propose to implement a hash table at each proxy/middlebox to avoid the need of multi-field policy matching

for most packets. The table stores $\langle f, a\rangle$ pairs, where f is a flow identifier (5-element tuple) and a is an action list. When a proxy/middlebox receives a packet with flow identifier f, it hashes f as index into the table to see if there is a matching pair $\langle f, a\rangle$. If there is one, a will be the action list. If there is no matching pair in the hash table, the proxy/middlebox resorts to the multi-field lookup in the policy table as is described previously to find the first matching policy $p = \langle d, a\rangle$. It then inserts $\langle f, a\rangle$ into the hash table so that the subsequent packets of the flow will find a quickly in the hash table, without having to perform expensive lookup in the policy table. Pairs stored in the hash table are soft-state, which will be timed out after a certain period absent of matching.

If a packet does not find a match in the hash table and the policy table, we insert $\langle f, null\rangle$ in the hash table so that the proxy/middlebox will find null for the subsequent packets of the flow, which will then be forwarded without further action — no need to consult with the policy table.

E. Avoid Packet Fragmentation

IP-over-IP adds an additional IP header to the packets, increases the packet length, and thus may result in packet fragmentation. One possible alternative is to modify the hash table for label switching, which helps reduce the need for IP-over-IP. This approach requires an additional label table at each middlebox.

Consider the first packet of a flow that is received by a policy proxy x. If the packet is not fragmented (which is generally true in properly configured networks [12]), as the proxy inserts $\langle f, a\rangle$ into its hash table, it adds an extra label field, l, which is locally unique in the table. The proxy inserts l into the unused fields in the packet header, such as the ToS byte and the fragmentation offset. It then forwards the packet (as payload) IP-over-IP to the next middlebox x'. When x' receives the packet, it also inserts an entry $\langle src \mid l, a\rangle$ into a label table, where '|' is the concatenation operator and a is the action list retrieved from its policy table using f. Middlebox x' then forwards the packet IP-over-IP to the next middlebox. When the packet reaches the last middlebox y specified in a, as the middlebox inserts an entry into its label table, it adds an extra field dst for the destination address of the packet. In other words, the last middlebox inserts $\langle src \mid l, a, dst\rangle$ in its label table before forwarding the packet. Moreover, it sends a control packet carrying f back to the proxy x; it knows the address of x because we can keep x as the source address in the outer IP header for all IP-over-IP tunnels from x all the way to y. When proxy x receives this control packet, it uses f to identify the entry $\langle f, a, l\rangle$ in its flow table and flags this entry for label switching.

For a subsequent, non-fragmented packet of flow f, as the proxy finds a matching entry $\langle f, a, l\rangle$ in its flow table, if the entry is not flagged for label switching, the same action as described in the previous subsection is performed. If the entry is flagged for label switching, the proxy will insert l into the packet header. It will then replace the destination address of the packet with the address of the next middlebox m_x^e and forward the packet, without an outer IP header as IP-over-IP will do.

When a middlebox x' receives a packet addressed to the middlebox without an outer IP header, it will use the label l and the source address src from the packet header to search the label table for a matching entry $\langle src \mid l, a\rangle$. It then replaces the destination address with the address of the next middlebox $m_{x'}^{e'}$ before forwarding the packet.

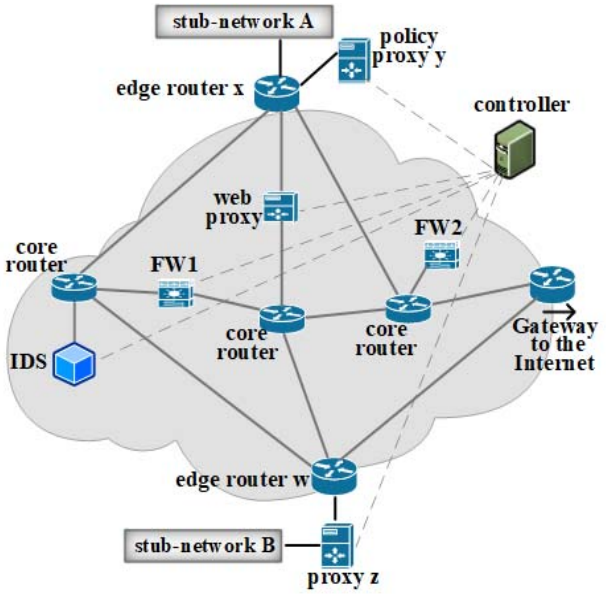
When the last middlebox in the action list a receives the packet, it searches its label table for a matching entry $\langle src \mid l, a, dst\rangle$ and replaces the destination address with dst before forwarding the packet.

The above approach switches from IP-over-IP to label switching after the first packet of a flow reaches its final middlebox, allowing the subsequent packets of the flow to be routed through a policy-enforced path without increasing packet length.

F. An Example

In Figure 3, we use an example to show how a network policy in Figure 3.a can be enforced with our design. The policy is to first forward web traffic from stub-network A to a web proxy. If the current version of pages requested is already cached, the request is honored. Otherwise, the web traffic is routed through the following sequence of middleboxes: Firewall $\rightarrow$ IDS. Let f be a flow from a host in stub-network A to a web server outside the network depicted in Figure 3. When *policy proxy y* receives the first packet of f, an associated action list, a, is retrieved from its policy table. In this case, a is $\{\text{web proxy}, \text{FW}, \text{IDS}\}$, where web proxy, FW1, and IDS are closest middleboxes implementing required network functions. The policy proxy embeds a locally unique label (1) into the header of the first packet of f before forwarding it (as a payload) IP-over-IP, with *proxy y* as the source, to the closest middlebox implementing the first network function in a (web proxy). As shown in Figure 3.b, $\langle f, a, 1, 0\rangle$ is also inserted in the policy proxy's hash table for quicker processing of subsequent packets of f.

If the requested page is not yet cached in the web proxy, then f will be forwarded to FW1. Once web proxy receives the first packet of f from *policy proxy y*, it retrieves f and the incoming label from the packet and a from its policy table. Afterwards, an entry $\langle src|1, a\rangle$ is inserted into its label table (Figure 3.d), where the source address of f is src. The packet is then tunneled to the nearest middlebox implementing the next network function in a (FW1) without changing the incoming source address and label on the outer header. The packet is similarly processed by subsequent middleboxes on its path until it reaches the last middlebox (see Figure 3.e). When the middlebox implementing the last



(a) user policy at the controller

Traffic	web traffic from stub-network A
Policy	web proxy → FW → IDS

(b) hash table at policy proxy y (after receiving 1st packet)

Flow	Action list	Label	Flag
f	a	1	0

(c) hash table at policy proxy y after receiving ack

Flow	Action list	Label	Flag
f	a	1	1

(d) label table at web proxy

key	Action list	Dest. address
src|1	a	-

(e) label table at FW1

key	Action list	Dest. address
src|1	a	-

(f) label table at IDS

key	Action list	Dest. address
src|1	a	dst

Figure 3: An example

network function in a (IDS) receives the packet, it inserts $\langle src|1, a, dst\rangle$ into its label table (Figure 3.f), where the destination address of f is dst. Henceforth, the packet is routed along a shortest path to its destination: IDS → core router → edge router w → Gateway.

At the same time, a control message is sent back to policy proxy y, which then turn on the label switching flag in its flow table (Figure 3.c). The control message is sent through a shortest path: IDS → core router → edge router x → proxy y. Consequently, subsequent packets of f are simply forwarded through the label switching path established above.

IV. Performance Evaluation

We use OMNET++ [13] to evaluate the performance of the proposed solution for automated policy enforcement in non-SDN networks. The prior work on automated policy enforcement [2], [4] relies on forwarding flexibility of software-defined switches. They can only be applied to SDN networks. In contrast, the proposed solution is designed solely for non-SDN networks, without any software-defined switches. Therefore, comparing with [2], [4] does not make sense. Because there is no prior art on automated policy enforcement based on software-defined middleboxes, we will use the hot-potato enforcement strategy (HP) and a randomized enforcement strategy (Rand) as the baselines to demonstrate the effectiveness of our load-balancing optimization.

A. Evaluation Settings

We implement the proposed solution on OMNET++ [13], which is a popular network simulation platform. In addition, we utilize the standard Internet stack, such as TCP, IPv4 OSPF, provided by the INET framework [14]. Our simulations use two network topologies. One is a real-world campus network topology, with two main gateways to the Internet, 16 core routers each connecting to both gateways and 10 edge routers. The second topology is randomly generated based on the Waxman model [15], in which the number of edge routers is set to 400, and the number of core routers is set to 25, each of which is connected to an equal number of edge routers. The core routers are interconnected based on the Waxman model [15], in which each router is assigned a pair of coordinates in a 100-by-100 region at random and the connection between two routers is probabilistically established with a distribution exponentially decreasing in their distance. The number of links from each core router to other core routers is set to 4. The forwarding paths in each network is determined by OSPF [5] using shortest-path routing.

We simulate three typical types of network policies: (1) many to one, (2) one to many, and (3) one to one. A many-to-one policy specifies a sequence of network functions to be applied on traffic from many source subnets to one destination subnet (or host). A one-to-many policy specifies a sequence of network functions to be applied on traffic from one subnet to many different destination. A one-to-one policy specifies a sequence of network functions to be applied on traffic from one subnet (or host) to another subnet (or host). More specifically in our simulations, for each many-to-one policy, we randomly choose a destination subnet (behind an edge router), use wildcard as the source, and consider an arbitrary service (i.e., destination port), with the action list being FW and IDS to protect the destination service from all external threats. For each one-to-many policy, we randomly choose a source subnet (behind an edge router), use wildcard as the destination, and consider

http traffic, with the action list being FW, IDS and WP for both security and web caching. Each such policy will have a many-to-one companion policy for the return web traffic. For each one-to-one policy, we randomly choose a pair of source subnet and destination subnet, with an arbitrary service and an action list being IDS and TM for intrusion detection and traffic measurement, as a means for the network admin to investigate the traffic between two subnets when malicious activities are suspected.

We apply four types of middleboxes in our simulation, implementing four functions: web proxing (WP), firewalling (FW), intrusion detection (IDS) and traffic measurement (TM). Their numbers are 4, 7, 7 and 4, respectively. The different numbers reflect their different frequencies of appearance in the policies. Each middlebox is connected to a randomly-chosen core router. In our load-balanced enforcement strategy, for each middlebox x that does not implement FW, we let M_x^{FW} contain 4 closest middleboxes that implement FW; for each middlebox x that does not implementing IDS, we let M_x^{IDS} contain 4 closest middleboxes that implement IDS; for each middlebox x that does not implement WP, we let M_x^{WP} contain 2 closest middleboxes that implement WP; for each middlebox x that does not implement TM, we let M_x^{TM} contains 2 closest middleboxes that implement TM.

We simulate one unit of time, during which the number of policy-matching flows ranges from 30000 to 300000, and their sizes follow a power law distribution in the range from 1 to 5000 packets. The total number of packets generated by these flows ranges from 1000000 to 10000000. These flows are randomly assigned to the three policy classes discussed above in the following way: one third to the one-to-many policy class (with the action list being FW $\rightarrow$ IDS), one third to the many-to-one policy class (with the action list being FW $\rightarrow$ IDS $\rightarrow$ WP), and one third to the one-to-one policy class (with the action list being IDS $\rightarrow$ TM).

B. Simulation Results

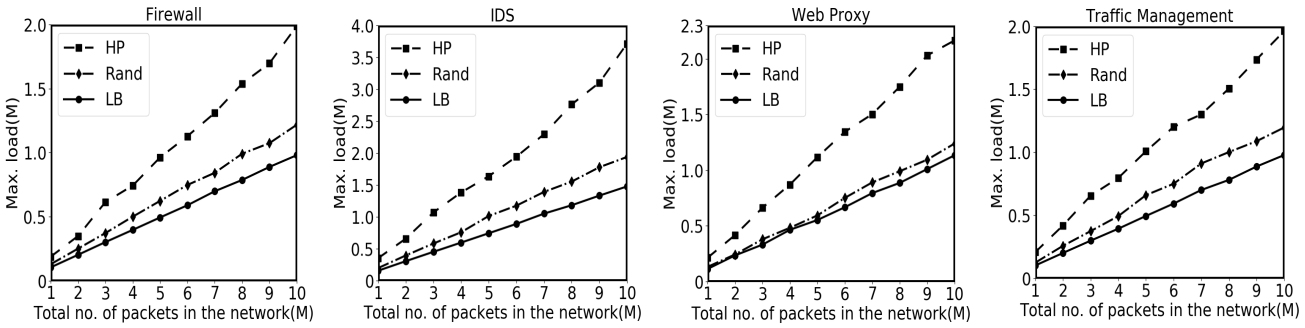
We first compare the hot-potato enforcement strategy (H-P), a randomized enforcement strategy (Rand) and the load-balanced enforcement strategy (LB) in terms of maximum load on a middlebox in each type. In hot-potato enforcement, each middlebox or proxy forwards all packets that require a certain function to the closest middlebox of that function type. In random enforcement, each middlebox or proxy x forwards a flow that needs a certain function e to a randomly chosen member of M_x^e. In load-balanced enforcement, each middlebox or proxy x forwards a flow that requires a certain function e to a middleboxes y selected from M_x^e with a probability proportional to $t_{e,p}(x, y)$, computed from Eq. 2.

The results on the university campus topology are shown in Fig. 4, where the four plots from left to right present the maximum loads on a firewall (FW), an intrusion detection system (IDS), a web proxy (WP), and a traffic measurement device (TM), respectively, in terms of number of packets in millions. In each plot, the horizontal axis represents the total traffic volume (in millions) of all flows in the network, and the vertical axis represents the maximum load (in millions) processed by a middlebox. In all four plots, the maximum loads increase linearly with traffic volume in the network, and as we expect, the load-balanced enforcement has a smaller maximum load than the random and hot-potato enforcement strategies. For example, in Figure 4(b), when the traffic volume is 5M, the maximum load on IDS is 1.66M packets under hot-potato enforcement, 1.01M under random enforcement and 0.74M under load-balanced enforcement. The results on the Waxman topology are shown in Figure 5. Similar conclusions can be made: In all four plots, the maximum loads increase linearly with traffic volume in the network, and the load-balanced enforcement has smaller maximum load than the random and hot-potato enforcement strategies

Table III shows the load distribution for each type of the middleboxes in the university campus topology. The table shows maximum and minimum loads for firewalls, intrusion detection systems, web proxies and traffic measurement devices. We can see that the load-balanced enforcement works much better than the hot-potato and random enforcement strategies. For example, the loads on FWs range from 0.4M to 1.89M under hot-potato and from 0.69M to 1.22M under random v.s. from 0.91M to 0.98M under load-balancing; the loads on IDSes range from 0.11M to 3.40M under hot-potato and from 0.93M to 1.99M under random v.s. from 1.37M to 1.47M under load-balancing; the loads on WPs range from 0.01M to 2.20M under hot-potato and from 0.45M to 1.24M under random v.s. from 0.47M to 1.10M under load-balancing; the loads on TMs range from 0.04M to 1.88M under hot-potato and from 0.44M to 1.23M under random v.s. from 0.51M to 0.98M under load balancing. The loads on WPs and TMs are less balanced because there are fewer numbers of them, allowing less flexiblity for balancing.

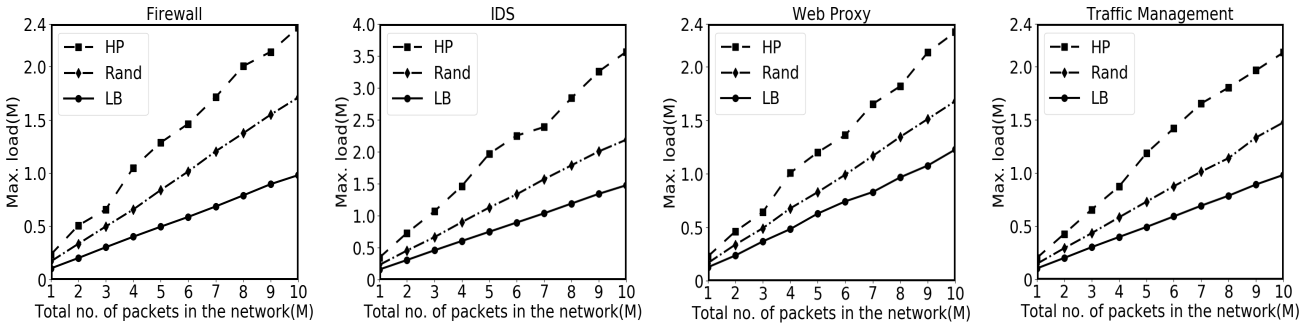
Table III: Load distribution in maximum and minimum loads (in number of packets) among middleboxes in the campus topology.

Middlebox	**Hot-potato (HP)**	**Random (Rand)**	**Load-balance (LB)**
FW max.	1891652	1223174	977257
FW min.	402753	687877	910051
IDS max.	3395230	1986925	1468925
IDS min.	106713	926704	1365438
WP max.	2203942	1235988	1105270
WP min.	12737	446230	464976
TM max.	1879304	1232254	978894
TM min.	44724	442673	511895



(a) Maximum load of an FW (b) Maximum load of an IDS (c) Maximum load of a WP (d) Maximum load of a TM

Figure 4: Comparison of maximum load on any middlebox in the campus topology



(a) Maximum load on an FW (b) Maximum load on an IDS (c) Maximum load of a WP (d) Maximum load of a TM

Figure 5: Comparison of maximum load on any middlebox in the Waxman topology

V. Related Work

End-to-End Explicit (E2E) Routing like Multi Protocol Label Switching (MPLS) [16] provides more efficient, flexible and robust traffic engineering mechanisms than the traditional IP network. However, due to a potentially large number of tunnels to be created and maintained, an exploding number of states needed to be stored in each switch, which will cause significant scalability issues when implementing a traffic engineering mechanism in MPLS based on Resource Reservation Protocol - Traffic Engineering (RSVP-TE) [17]. Our work does not store any states in the underlying switches/routers. Rather, network policy enforcement routing information are stored compactly in middelboxes/policy proxies through software for quick retrieval.

Another source routing architecture similar to our work is Segment Routing (SR) [18]. In SR framework, packets are routed through paths embedded in their header as an ordered list of segments. Segments can be "follow the shortest path to a host, switch or router", "switch to a port or link" or "apply a given service or policy". Traffic engineering requires one or more labels be encoded in packets. Strict encoding, which involves encoding segment identifier (SID) of all the hops and services on a packet path in its header, may increase the packet overhead and violate the hardware limitation to the length of segment lists [19], [20]. One of our design goals is to avoid increasing packets length to prevent fragmentation.

We do not manipulate network functionalities of each middlebox like [1], but simply steers packets through sequences of middleboxes. PLayer [21] forward packets to unmodified middleboxes through policy-aware switches. Other SDN-based network policies enforcement schemes use tagged tunnels to route packets through sequences of middleboxes [2], [4]. Our work is deployable on the traditional IP networks with no restriction on the placement of middleboxes. Through a software-defined data structure, our work is able to store much more states information than flow table in SDN. In addition, only select network devices (polixy proxies and middleboxs) are connected to the controller, compare to all underlying switches in SDN. These greatly enhance scalability of our work.

VI. Conclusion

This paper proposes a new network policy enforcement architecture by introducing software-defined middleboxes that can be directly configured by a centralized controller. Unlike SDN controllers, the middlebox controller is not

involved in assisting packet forwarding at flow level. It only configures the middleboxes based on user-specified policies and traffic measurement, and thus is unlikely to become a bottleneck. We propose a lightweight hot-potato enforcement strategy and an optimized load-balanced policy enforcement. We enhance them with middlebox-based flow table and label switching to relieve processing overhead and avoid packet fragmentation. We use OMNET++ simulation to demonstrate the effectiveness of the proposed solution with load-balancing optimization.

ACKNOWLEDGMENT

This work was supported by National Science Foundation of US under grant CNS-1719222.

REFERENCES

[1] Z. G. A. Gember, P. Prabhu and A. Akella, "Toward software-defined middlebox networking," in *Proceedings of the HotNets-XI*. IEEE, 2012.

[2] Z. A. Qazi, C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," in *Proceedings of ACM SIG-COMM*. ACM, 2013.

[3] W. Z. M. C. A. Khurshid, X. Zou and P. B. Godfrey, "Veriflow: Verifying network-wide invariants in real time," in *Proceedings of the HotSDN*, 2012.

[4] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, 2014.

[5] J. Moy, "Ospf version 2," *Internet Request For Comments RFC 1247*, July 1991.

[6] J. Smith. Introduction to eigrp. [Online]. Available: https://www.cisco.com/c/en/us/support/docs/ip/enhanced-interior-gateway-routing-protocol-eigrp/13669-1.html

[7] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 6th ed. Pearson, 2012.

[8] P. Gupta and N. McKcown, "Packet classification on multiple fields," in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, vol. 29. ACM, 1999, pp. 147–160.

[9] D. E. TAYLOR, "Survey and taxonomy of packet classification techniques," *ACM Computing Surveys (CSUR)*, vol. 37, no. 3, pp. 238–275, Sep. 2005.

[10] B. Y. Y. X. Y. Qi, L. Xu and J. Li, "Packet classification algorithms: From theory to practice," in *Proceedings of the INFOCOM*. IEEE, 2009.

[11] D. P. Mehta and S. Sahni, *Handbook of Data Structures and Applicationsh*. Chapman and Hall/CRC, 2018.

[12] D. M. C. Shannon and k. Claffy, "Characteristics of fragmented ip traffic on internet links," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet measurement (IMW)*. ACM, 2001, pp. 83–97.

[13] Omnet++ discrete event simulator. (2019). [Online]. Available: https://omnetpp.org/

[14] Inet framework. (2019). [Online]. Available: https://inet.omnetpp.org/

[15] B. M. WAXMAN, "Routing of multipoint connections," *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, vol. 6, no. 9, December 1988.

[16] I. Minei *et al.*, *MPLS-enabled applications: emerging developments and new technologies*. John Wiley & Sons, 2010.

[17] A. Cianfrani, M. Listanti, and M. Polverini, "Incremental deployment of segment routing into an isp network: a traffic engineering perspective," *in IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 3146–3160, Oct. 2017.

[18] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The Segment Routing Architecture," *Proc. of IEEE GlobeCom*, 2015.

[19] A. Giorgetti, P. Castoldi, F. Cugini, J. Nijhof, F. Lazzeri, and G. Bruno, "Path encoding in segment routing," in *Proceedings of the Global Communications Conference (GLOBECOM)*. IEEE, 2015, pp. 1–6.

[20] R. Guedrez, O. Dugeon, S. Lahoud, and G. Texier, "Label encoding algorithm for mpls segment routing," in *Proceedings of the 15th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2016, pp. 113–117.

[21] A. T. D. A. Joseph and I. Stoica, "A policy-aware switching layer for data centers," in *Proceedings of SIGCOMM*. IEEE, 2008.