Scalable and Balanced Policy Enforcement Through Hybrid SDN-Label Switching

Olufemi Odegbile* Shigang Chen† Youlin Zhang‡
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, Florida, USA
Email: *oodegbile, †sgchen, ‡ylzh10@ufl.edu

***Abstract*—Software-defined networks facilitate automatic policy enforcement with dynamic routing of flows through a sequence of middleboxes that offer the required network functions. As a result, network policy enforcement based on middleboxes, which is tedious and error-prone to perform in traditional IP networks, is greatly simplified. However, TCAM-based flow tables in SDN are small and energy-demanding, which limits the scalability of policy enforcement. This paper proposes a hybrid SDN-label switching scheme that combines TCAM-based switching (in SDN) at the network edge with label switching in the network core to provide scalable policy enforcement without compromising per-flow management capability. A linear optimization is proposed to balance workloads among the middleboxes. We demonstrate on OMNET++ that our proposed solution incurs much smaller processing/communication overhead and achieves better load-balancing when comparing with the prior art.**

I. Introduction

Essential to network management is the enforcement of various network policies. For example, network administrators may want to restrict access of blacklisted addresses to critical infrastructures, reduce rates of clients whose total traffic exceed a threshold, or implement alternative paths to the shortest paths for large flows. One way for policy enforcement is to use middleboxes that offer critical services such as firewalling, intrusion detection, proxing, load balancing and traffic measurement [1]. However, tedious manual placement and configuration of middleboxes are error-prone and unable to dynamically respond to changes in the networks [2]. Network administrators must therefore deploy an efficient, accurate and scalable enforcement strategy.

Software-defined networks (SDN) provide a flexible approach of enforcing network-wide policies. Through a centralized architecture, SDN switches can be configured to steer packets through a sequence of middleboxes irrespective of their placement within the network. In addition, the SDN controller can dynamically respond to changes in the network. For example, the controller can reconfigure switches with new rules to reroute large flows so as to ease traffic congestion. However, while traditional IP networks are incredibly scalable, SDN scales poorly due to limited TCAMs available to store flow rules [3]–[6]. The memory requirement of SDN is further exacerbated because flow rules are not only used for routing but also policy enforcement. As a result, scalability is a major issue in large datacenters, which require a large number of rules to enforce their network policies [7].

There are several prior solutions for improving SDN scalability. Flows can be aggregated with wildcard rules to efficiently utilize flow tables. However, this approach can still require a large number of flow rules and is unsuitable for per-flow policy enforcement [5]. In another solution based on source routing [8], the controller configures ingress switches to embed a path into each incoming packet's header. Subsequently, interior switches simply forward packets based on the routing information contained in their headers. Source routing increases packet length by a variable amount dependent on path length. It degrades routing performance and may cause packet fragmentation. Besides, there is a hardware limitation to the allowable segment length (Maximum Segment Length Depth) [9] and the optimal path encoding problem is an *APX-hard* problem [10].

This paper attempts to address the TCAM-induced scalability problem of SDN without losing the flexibility of fine-grained per-flow policy enforcement or causing performance degradation and packet fragmentation. We observe that the scalability issue is most serious in the network core where all flows from the network edge converge towards and traverse through. Our solution follows a decades-long network design principle: pushing network complexities to the edge and keeping the core simple [11], [12]. The idea is that if TCAM is too small and unscalable in the core, we shall replace it with something else that is scalable and can be implemented in SRAM or other types of memory, while the flexibility of SDN is still retained at the edge. To this end, we propose a hybrid SDN-label switching design, which keeps TCAM-based flow tables at edge switches and uses label switching at core switches, both of which are configured by the controller. In our new design, the transition from SDN switching to label switching happens at the edge SDN-capable switches. Label switching can be efficiently implemented by a label-indexed hash table. Note that core switches may be label-switching capable only or SDN-label capable in which case TCAM-based flow tables can be used for network functions (such as traffic engineering) other than policy enforcement considered in this paper.

Tunnelling between middleboxes in [2], [13] can help reduce the number of flow entries for each flow, but their method still requires TCAM-based flow table entries in the network core, in particular at the switches that directly connect to the middleboxes, which contrasts to our proposed solution that eliminates the need of TCAM-based flow tables in the core and is therefore more scalable.

The contributions of this paper are summarized as follows: We propose a new hybrid SDN-label switching architecture, which uses label switching to achieve scalability at the network core, while using SDN switching for flow identification and label assignment at the edge. We present a load-balanced design for per-flow policy enforcement which routes packets to a series of required middleboxes subject to the middlebox capacity constraints. Finally, we evaluate the scalability of our design and the effectiveness of our load-balanced enforcement method on OMNET++ [14].

II. NETWORK MODEL AND PROBLEM STATEMENT

A. Network Model

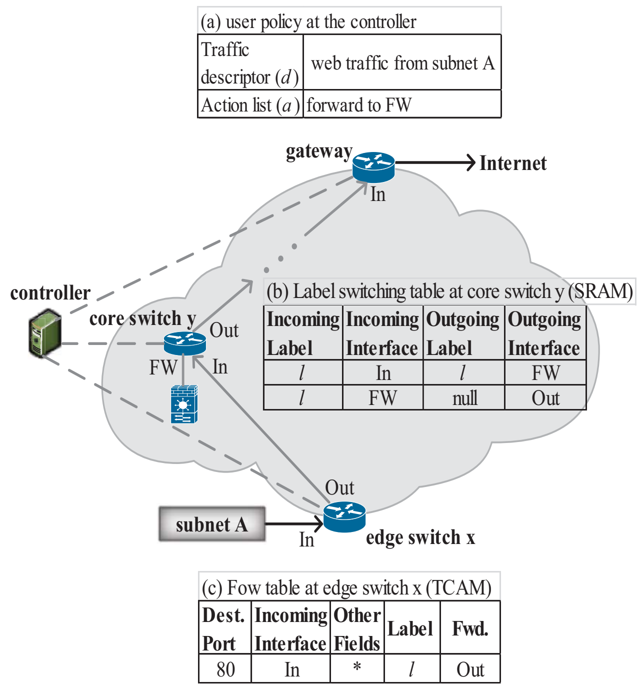
A large SDN-capable enterprise network consists of a centralized controller, SDN-capable edge switches each directly connected to a subnet, label-switching core switches that interconnect the edge switches, and middleboxes that are deployed in the core (each connecting to a core switch at a chosen location) to implement various required network functions such as firewalling, proxying and intrusion detection. The controller has the complete knowledge of the network topology, collect measurements from the switches, determine the policy enforcement paths for all flows, and configure both the edge switches and the core switches for their SDN flow tables (based on openFlow [15]) and label-switching tables, respectively. Refer to Figure 1 for an example, where the flow table and the label-switching table will be explained later.

Because SDN switching is optional in the core and label switching is intended for scalable policy enforcement, we need a routing structure for traffic that does not match any policy. For this purpose, we adopt the approach of [6] that integrates traditional routing into SDN networks for better scalability. We stress that the work [6] does not consider policy enforcement or label switching, and thus it is orthogonal to our work. Following [6], the switches run a traditional routing protocol such as OSPF [16] to provide the default shortest paths using the traditional routing tables, while flow tables (for SDN-capable switches) are used for traffic engineering as in [6] and label-switching tables are used in the core for policy enforcement.

B. Problem Definition

A network policy p is defined as an ordered pair $\langle d, a \rangle$, where d is a traffic descriptor (identifying flows that match this policy) and a is an ordered action list of required network functions. For example, d may specify a source subnet prefix, a destination subnet prefix, and a destination port, with other fields being wildcards by default, and a may be firewalling, which requires all matching flows to go through a middlebox that implements the firewall function.

Given an arbitrary set of network policies defined by users, the problem is how to configure all edge/core switches such that any flow that matches a certain policy $\langle d, a \rangle$ will be routed



(a) user policy at the controller

Traffic descriptor (d)	web traffic from subnet A
Action list (a)	forward to FW

(b) Label switching table at core switch y (SRAM)

Incoming Label	Incoming Interface	Outgoing Label	Outgoing Interface
l	In	l	FW
l	FW	null	Out

(c) Fow table at edge switch x (TCAM)

Dest. Port	Incoming Interface	Other Fields	Label	Fwd.
80	In	*	l	Out

Fig. 1: Architecture of hybrid SDN-label Switching (HSLS).

through a series of middleboxes that implement the network functions in the exact order specified by the action list a. If there are multiple matching policies, we take the first one.

III. Network Policy Enforcement by Hybrid SDN-label Switching

A. Design of Hybrid SDN-label Switching

Our primary design goal is to push network policy enforcement decisions to edge switches and the controller, while leaving interior network simple and mostly to forward packets. To this end, we replace flow-table forwarding (from SDN) with lightweight label switching in the network interior. In particular, the core switches use *label-switching tables* (LST), which are proactively configured by the controller, to route packets through a series of required middleboxes and then to the destinations. As shown in Figure 1, LST has four fields compared to a much larger number of fields in SDN's flow table. The four fields are Incoming Label, Incoming Interface, Outgoing Label, and Outgoing Interface. In contrast, the edge switches still use the flow tables of SDN, with one extra *label* field.

For each switch, the controller pre-computes a certain number of alternative enforcement paths for each policy that can be applied to traffic through the switch. For example, consider an arbitrary policy $\langle d, a \rangle$, where d overlaps with traffic through an edge switch x. The controller may first find the shortest enforcement path through a series of middleboxes as specified in a by a modified Dijkstra's algorithm. It then removes those middleboxes, and tries to find another shortest path that does not overlap with the first one. This process repeats until no more enforcement path can be found. Multiple paths are used for the purpose of load balancing. We will refer to the shortest enforcement path through any series of middleboxes as specified in a as the *shortest policy path* (SPP). The controller will assign a different label to each enforcement path for this policy at switch x. It will also instruct all core switches along each enforcement path to configure their label-switching tables by adding proper entries so that the received packets with the matching incoming label will be forwarded correctly to the next hop on the path.

When the edge switch x receives the first packet of a flow f, it will send a *packetIn* massage to the controller, which searches for a matching policy. If there is one, the controller will look up for the pre-computed enforcement paths of the policy, select one path (by hashing the flow ID to one of the paths), and send the corresponding label in a *flowMod* message back to x. The edge switch x will create a flow-table entry with the received label for f. All subsequent packets of f will match this entry, and switch x will embed the label of the entry in the packets, either by inserting a label header or piggybacking the label in the unused header fields (such as such as TOS byte and fragmentation offset if the network is configured to avoid fragmentation).

For flows that do not match any policy, they may be routed through pre-established default paths, or the controller may dynamically allocate a (shortest or load-balanced) path and configure the edge switch and the core switches along the path with dynamically assigned labels.

When a core switch y receives a packet of f, it extracts the label from the packet header and looks up in its LST to retrieve an outgoing label and an outgoing network interface. The packet will be forwarded to the outgoing interface after the label carried in its header is substituted by the outgoing label. Once the last network function applicable to the packet is enforced, the label embedded in its header is removed. Henceforth, the packet is routed along the default routing path provided by the traditional routing protocol.

At the core switches, their label-switch tables are indexed by labels and can be implemented as hash tables in SRAM or other types of memory. At the edge switches, the traditional flow tables perform multi-field matching as specified in OpenFlow, which requires TCAM.

B. Load-Balancing Policy Enforcement

As discussed in Section III-A, for each policy, the controller pre-computes and deploys a number of alternative enforcement paths from each edge switch whose traffic overlaps with the policy. When a new matching flow arrives at the edge switch, we can randomly choose one from the alternative paths for enforcement. This simple method does not consider traffic dynamics and middlebox capacities, and it may result in unbalanced load distribution among the middleboxes, overloading some while underutilizing others. To solve this problem, we introduce a load-balanced path selection method. Below we first give the notations in our formulation and then present the optimization.

Let E be the set of edge switches, $e \in E$ refer to an edge switch, P be the set of policies, $p \in P$ refer to a policy, $P_e \subset P$ be the set of policies that overlap with traffic through edge switch e, and $H_{e,p}$ be the set of alternative enforcement paths for a policy p at an edge switch e. Clearly, all paths in $H_{e,p}$ start from e. Let M be the set of middleboxes, $m \in M$ refer to a middlebox, and $c(m)$ be the capacity of m.

Let $F_{e,p}$ be the set of all flows received by e that match $p \in P_e$, $f \in F_{e,p}$ refer to a flow, $T_{e,p}$ be the total traffic volume of all flows in $F_{e,p}$ during the most recent measurement period, and T_f be the traffic volume of a flow f in the most recent period. The values of $T_{e,p}$ and T_f, $\forall f \in F_{e,p}$, are measured by edge switch e and reported to the controller. Let $t(h_{e,p})$ be the portion of traffic $T_{e,p}$ that should be routed through an enforcement path $h_{e,p} \in H_{e,p}$ in order to achieve load balancing among the middleboxes. We will compute (and then enforce) the optimal traffic distributions $t(h_{e,p}), \forall e \in E, p \in P_e, h_{e,p} \in H_{e,p}$, by solving the following load-balancing linear programming optimization.

$$\begin{aligned}
&\min \quad \lambda \\
&\text{s.t.} \\
&\textstyle\sum_{h_{e,p} \in H_{e,p}} t(h_{e,p}) = \quad T_{e,p}, \quad \forall e \in E, p \in P_e \\
&\textstyle\sum_{e \in E, p \in P_e} \sum_{h_{e,p} \in H_{e,p}: m \in h_{e,p}} t(h_{e,p}) \\
&\qquad \leq \lambda \cdot c(m), \forall m \in M \\
&t(h_{e,p}) \qquad \geq 0, \quad \forall e \in E, p \in P_e, h_{e,p} \in H_{e,p} \\
&\lambda \ \leq 1
\end{aligned} \tag{1}$$

The first constraint ensures that traffic portions on all alternative enforcement paths sum up to the total traffic volume expected. The second constraint ensures that the aggregate traffic load on each middlebox does not exceed its capacity, where λ is the largest load factor among all middleboxes, which we will minimize. Eq. 1, being a linear optimization problem, can be solve in polynomial time.

After $t(h_{e,p}), \forall e \in E, p \in P_e, h_{e,p} \in H_{e,p}$, are determined, the controller calculates a weight for each path in $H_{e,p}$ as

$$w(h_{e,p}) = \frac{t(h_{e,p})}{\sum_{h_{e,p} \in H_{e,p}} t(h_{e,p})}. \tag{2}$$

The above weights are essentially the normalized traffic portions, such that they add to one, i.e., $\sum_{h_{e,p} \in H_{e,p}} w(h_{e,p}) = 1$. The controller wants to make sure that the traffic at e matching p is distributed among the paths in $H_{e,p}$ in proportion to the $h_{e,p}$ values, i.e., in proportion to the weights $w(h_{e,p})$. Since the weights are summed to one, for each new flow arriving at e matching p, the controller will simply select a path from $H_{e,p}$ with a probability of $w(h_{e,p})$ for each path $h_{e,p}$. This can be easily implemented by hashing the flow ID to a random number in $[0, 1]$ and checking which segment it falls as the weights $w(h_{e,p})$ divide $[0, 1]$ into $|H_{e,p}|$ consecutive segments.

C. An Example

We illustrate our load-balanced policy enforcement method in Figure 2. We show how a network policy p in Figure

(f) label switching table at c_3 (SRAM)

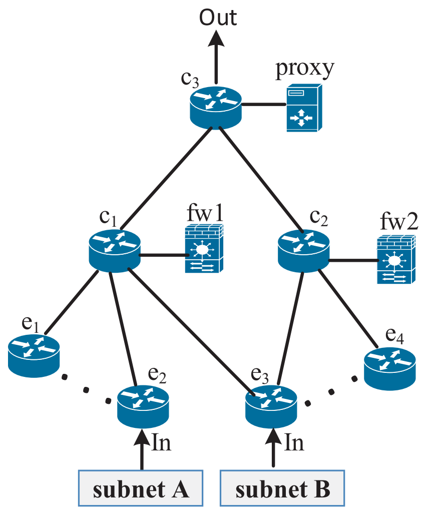
Incoming Label	Incoming Interface	Outgoing Label	Outgoing Interface
l_3	c_1, c_2	l_3	proxy
l_3	proxy	null	Out

(e) label switching table at c_2 (SRAM)

Incoming Label	Incoming Interface	Outgoing Label	Outgoing Interface
l_2	e_3	l_2	fw2
l_2	fw2	l_3	c_3

(d) label switching table at c_1 (SRAM)

Incoming Label	Incoming Interface	Outgoing Label	Outgoing Interface
l_1	e_2	l_1	fw1
l_1	fw1	l_3	c_3



Controller

(a) user policy **p**

Traffic descriptor (d)	web traffic from subnet B
Action list (a)	fw → proxy

(b) possible enforcement paths for **p**

first path			second path		
weight = 0.3	c_1	l_1	weight = 0.7	c_2	l_2

(c) flow table at e_3 (TCAM)

Dest. Port	Incoming Interface	Other Fields	Label	Fwd.
80	In	*	l_2	c_2

Fig. 2: An example of policy enforcement by hybrid SDN-label switching

2.a can be enforced with our method. First, the controller pre-computes two possible enforcement paths for p. Each enforcement path is characterized by its weight (i.e., probability of being chosen), outgoing interface and path label. After executing Eq. 1 & 2, the weights of the paths are 0.3 and 0.7, respectively; see Figure 2.b. This implies that 30% of the traffic from subnet B that matches p (web traffic) will be routed along the first enforcement path while the remaining 70% will be routed along the second enforcement path.

Assume that the controller probabilistically selects the second enforcement path. Then edge switch e_3 is configured to insert l_1 in the header of a packet from subnet B that matches p before forwarding the packet to core switch c_2 (see Figure 2.c). At the same time, c_2 is configured to first forward the packet to firewall fw2 (see the first entry of Figure 2.e) and then switch its label to l_3 before forwarding the packet to core switch c_3 (see the second entry of Figure 2.e). Similarly, c_3 is configured to first forward the packet to the proxy (see the first table entry of Figure 2.f) and then remove its label before routing the packet along the default routing path provided by a traditional routing protocol (see the second entry of Figure 2.f).

IV. Performance Evaluation

In this section we perform numerical evaluation of the proposed policy enforcement solution and compare it with the prior art.

A. Evaluation Settings

Our evaluation is performed with OMNET++ [14] and INET [17], which provides the standard internet protocol stacks. First, we compare our *hybrid* SDN-label switching method on policy enforcement with the *hop-to-hop* method and the *tunnel* method in [2]. To remove the restriction imposed by the TCAM-based flow table of limited size, the hybrid method replaces it with label switching in the core. For the hop-by-hop and tunnel methods, we set the size to 1500 entries, following [2], [6]. When a flow table is overflown, we need to remove some existing rules (which may be done based on a policy such as least-used first) to make room for new ones, which generates additional overhead between switches and the controller when packets from the removed flows show up and their flow rules have to be put back. This overhead is what we will compare. Second, we compare our *load-balancing* (LB) path selection method with two alternative methods: 1) the *single* path method that puts all flows matching a policy at a switch on the same shortest enforcement path, and 2) the *random* path selection method that randomly chooses one from the pre-computed alternative enforcement paths.

Our evaluation uses two topologies. The first real topology is based on a campus backbone topology, which comprises two main core routers connecting 16 core routers. Each core router is connected to 10 edge routers. The second topology is randomly generated based on the Waxman model [18]. It comprises 25 core routers interconnected based on the Waxman model [18]. Similarly, each core routers is connected to 10 edge routers.

We consider four types of network functions: firewalling (FW), intrusion detection (IDS), web proxing (WP) and traffic measurement (TM). The number of middleboxes offering these network functions are 8, 8, 4 and 4, respectively, which reflect their different frequencies of appearance in the policies we enforce with our solution. Each middlebox is randomly connected to a core router.

Our experiments use three types of policies. Each first-type policy matches a certain popular external web address and requires all http traffic towards the destination to go through FW → IDS → Proxy. It is assigned 32 alternative enforcement paths going through half middleboxes of each type, i.e., $4 \times 4 \times 2 = 32$. Each second-type policy matches a certain external address under security watch and all inbound flows from that address goes through FW $\rightarrow$ IDS. It is assigned 16 alternative enforcement paths. Each third-type policy matches a specific pair of source/destination addresses under bandwidth watch and their flows are sent through IDS $\rightarrow$ TM. It is assigned 8 alternative enforcement paths.

In each evaluation run, 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 in these flows ranges from 1000000 to 10000000. These flows are randomly assigned to the three policy types, each having one third of the flows.

B. Evaluation Results

First, we evaluate the proposed *hybrid* method with the *hop-to-hop* and *tunnel* methods on processing/communication overhead in terms of the total number of requests for the controller to set up flow-table entries (rules) for unmatched arrival packets. This overhead reflects the scalability issue caused by limited flow-table size. Table I compares the overheads of the three methods on the Waxman network. The hybrid method has much smaller overhead than the other methods. For example, when the total traffic volume is 5000000 packets, the hybrid method generated less that 20K requests (from edge switches), compared to approximately 200K and 130K requests by *hop-to-hop* and *tunnel* respectively. We omit the results on the campus network due to space limitation.

Second, we evaluate the *load-balancing* method with the *single* and *random* methods on the maximum middlebox load. The results are presented in Figure 3 and Figure 4 based on the campus network and the Waxman network, respectively. The four plots from left to right show the maximum loads on a firewall (FW), an intrusion detection system (IDS), a web proxy (WP), and a traffic measurement device (TM), respectively. 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 the *load-balanced* method (LB) has a smaller maximum load than the *single* and *random* methods. For example, in Fig. 3(a), when the traffic volume is 10M, the maximum load on FW is 1.74M packets for the *single* method, 1.13M for the *random* method and 0.87M for LB. Similarly, from the results on the Waxman network in Fig. 4(a), when the traffic volume is 10M, the maximum load on FW is 2.54M packets for the *single* method, 1.13M for the *random* method, and 0.87M for LB.

Table II shows the load distribution for each type of middleboxes in the campus topology. The table shows maximum and minimum loads for firewalls, intrusion detection systems, web proxies, and traffic measurement devices, when the traffic volume is 10M packets. In our evaluation, the capacities of the four type of middleboxes are 1M, 1.5M, 1M and 1M respectively. We can see that the *load-balanced* method performs better than the *single* and *random* methods. For example, the loads on FWs range from 0M to 2.54M under *single* method and from 0.53M to 1.13M under *random* method v.s. from 0.75M to 0.87M under LB; the loads on IDSes range from 0M to 3.69M under *single* method and from 0.75M to 1.77M under *random* method v.s. from 1.21M to 1.29M under LB; the loads on WPs range from 0M to 2.06M under *single* method and from 0.48M to 1.18M under *random* method v.s. from 0.78M to 0.87M under LB; the loads on TMs range from 0M to 2.16M under *single* method and from 0.47M to 1.19M under *random* method v.s. from 0.79M to 0.87M under LB. In general, the *single* and *random* methods overload some middleboxes while others are under-utilized; the LB method overloads no middlebox.

TABLE I: Comparison of processing/communication overheads (rounded to the nearest thousand) of *hop-to-hop*, *tunneling* and *hybrid* schemes.

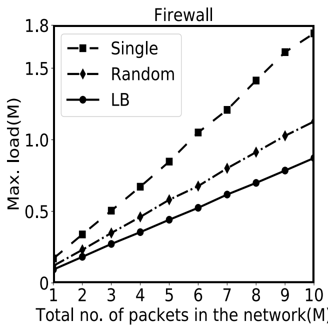
Total number of packets in the network	**Hop-by-hop**	**Tunnel**	**Hybrid**
500000	21000	5000	2000
1000000	110000	36000	4000
1500000	245000	108000	6000
2000000	422000	208000	8000
2500000	624000	339000	10000
3000000	874000	485000	12000
3500000	1148000	657000	14000
4000000	1423000	838000	16000
4500000	1718000	1032000	18000
5000000	2028000	1274000	20000

TABLE II: Load distribution in maximum and minimum loads (in number of packets) among middleboxes in a Waxman topology.

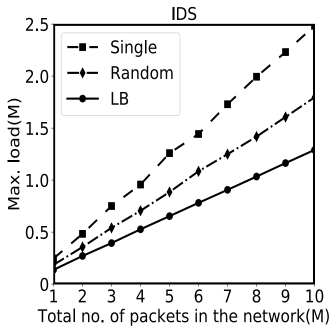
Middlebox	**Single**	**Random**	**Load-balancing (LB)**
FW max.	2542757	1125896	872828
FW min.	0	537235	747647
IDS max.	3693028	1768889	1286771
IDS min.	0	750334	1212795
WP max.	2058928	1183453	870169
WP min.	0	484221	777781
TM max.	2163695	1194458	869870
TM min.	0	470034	786199

V. Conclusion

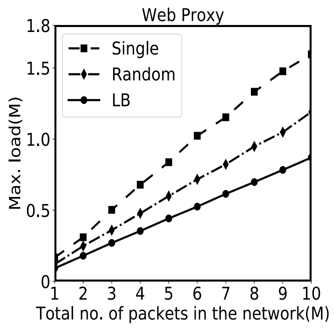
This paper proposes a new network policy enforcement architecture by introducing a *hybrid* SDN-label switching. Unlike the tradition software-defined newtorks (SDN), whose scalability is limited by the TCAM-based forwarding scheme in the network core, our hybrid architecture uses a highly scalable label switching in the network core to enforce network-wide policies. As a result, the scalability of our *hybrid* SDN-label switching is enhanced while still capable of fine-grained policy management. We formulate a linear optimization for



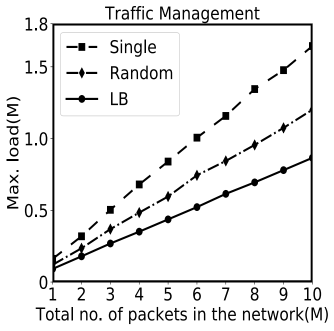
(a) Maximum load of a FW



(b) Maximum load of a IDS

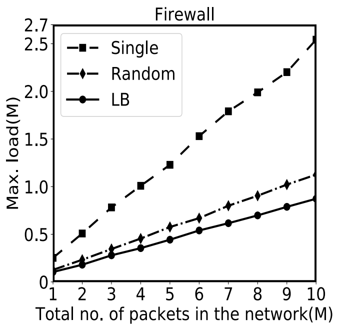


(c) Maximum load of a WP

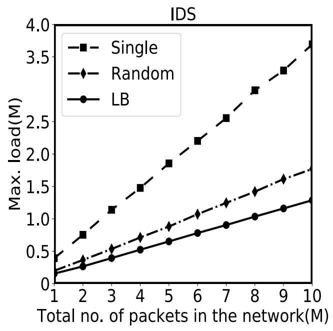


(d) Maximum load of a TM

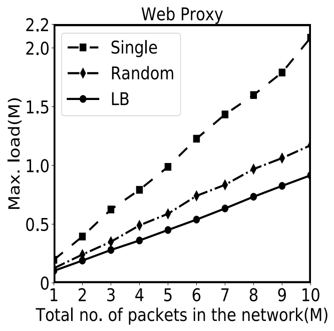
Fig. 3: Comparison of maximum load on any middlebox in the campus topology



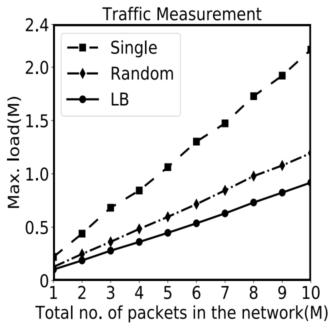
(a) Maximum load of a FW



(b) Maximum load of a IDS



(c) Maximum load of a WP



(d) Maximum load of a TM

Fig. 4: Comparison of maximum load on any middlebox in a Waxman topology

balance traffic distribution among the middleboxes. Through OMNET++ simulation, we demonstrate that our scheme generates far fewer processing/communication overheads than prior arts and the effectiveness of our load-balancing method.

Acknowledgment

This work was supported by National Science Foundation of US under grants CNS-1719222 and STC-1562485.

References

[1] V. Sekar, S. P. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi, "The middlebox manifesto: enabling innovation in middlebox deployment," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks (HotNets-X)*. ACM, 2011.

[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 SIGCOMM*. ACM, 2013.

[3] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," in *Proceedings of ACM SIGCOMM*, vol. 41, no. 4. ACM, 2011.

[4] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," in *Proceedings of ACM SIGCOMM*. ACM, 2010.

[5] Y. Zhang, "An adaptive flow counting method for anomaly detection in sdn," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 2013, pp. 25–30.

[6] H. Xu, H. Huang, S. Chen, and G. Zhao, "Scalable software-defined networking through hybrid switching," in *Proceedings of IEEE INFOCOM*. IEEE, 2017.

[7] M. Moshref, M. Yu, A. Sharma, and R. Govindan, "vcrib: Virtualized rule management in the cloud," in *Conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 157–170.

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

[9] 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.

[10] A. Hari, U. Niesen, and G. Wilfong, "On the problem of optimal path encoding for software-defined networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 1, pp. 189–198, Feb. 2017.

[11] D. D. Clark, *Designing an Internet*, 1st ed., ser. Information Policy Series. The MIT Press, 2018.

[12] R. Bush and D. Meyer, "Some internet architectural guidelines and philosophy," *Request For Comments RFC 3439*, December 2002.

[13] 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.

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

[15] Openflow switch specification v1.5.1. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

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

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

[18] B. M. Waxman, "Routing of multipoint connections," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, December 1988.