Shadow MACs: Scalable Label-switching for Commodity Ethernet

Kanak Agarwal
IBM Research
Austin, TX, USA
kba@us.ibm.com

Colin Dixon[*]
Brocade
San Jose, CA, USA
colin@colindixon.com

Eric Rozner
IBM Research
Austin, TX, USA
erozner@us.ibm.com

John Carter
IBM Research
Austin, TX, USA
retrac@us.ibm.com

ABSTRACT

While SDN promises fine-grained, dynamic control of the network, in practice limited switch TCAM rule space restricts most forwarding to be coarse-grained. As an alternative, we demonstrate that using destination MAC addresses as opaque forwarding labels allows an SDN controller to leverage large MAC (L2) forwarding tables to manage a plethora of fine-grained paths. In this *shadow MAC* model, the SDN controller can install MAC rewrite rules at the network edge to guide traffic on to intelligently selected paths to balance traffic, avoid failed links, or route flows through middleboxes. Further, by decoupling the network edge from the core, we address many other problems with SDN, including consistent network updates, fast rerouting, and multipathing with end-to-end control.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

Keywords

routing; ethernet; label-switching; consistent updates; software-defined networking; datacenter

1. INTRODUCTION

Software-defined networking (SDN) promises fine-grained control of the network both spatially and temporally. For example, OpenFlow [29] enables forwarding decisions to be made based on nearly any packet header field, in contrast to traditional coarse-grained destination-based forwarding. Further, forwarding rules can be updated hundreds of times per second [34] based on network-wide measurements gathered in milliseconds [31]. SDN applications leverage these features to implement near-optimal traffic engineering [8, 14], load balancers [39], and DDoS mitigation [41].

[*]This work completed at IBM.

HotSDN'14, August 22, 2014, Chicago, IL, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2989-7/14/08 ...$15.00.
http://dx.doi.org/10.1145/2620728.2620758.

Unfortunately, current hardware limitations make straightforward implementation of fine-grained spatial control problematic. Current switches have only a few thousand flexible rules [37], making it hard to provide even host-pair granularity for forwarding, so forwarding is typically based on aggregates [16, 26]. Prior work to leverage large L2 forwarding tables [37] avoids routing on aggregates, but limits forwarding to be destination-MAC-based.

Further, while individual rules can be installed hundreds of times per second, frequent path updates can have unintended consequences [23, 32]. While rules are being updated from one configuration to another, the network may pass through intermediary states that violate policy, cause congestion, form loops, or drop traffic.

We believe these limitations have contributed significantly to the fact that most real-world SDN deployments are based on overlays [22]. Hardware SDN deployments are typically small proofs of concept. Issues around consistent updates and rule space exhaustion tend to only arise at scale, so small proof of concept deployments do not observe or address these issues.

In this paper, we note label switching solves both problems. Encoding fine-grained paths as labels allows (almost) all forwarding decisions to be made using fixed-width, exact-match lookups, which map easily into large simple hardware tables without requiring large expensive TCAM tables. Further, since these paths use opaque labels, they can be installed in the network before any traffic is directed to them, which ensures per-packet consistency [32]

A recent position paper by Casado et al. [11] suggests that next-generation networks should combine an intelligent network edge with a label-switched core. Casado et al. arrive at this conclusion based on separating the concerns of end points, switches, and operators. We arrive at the same conclusion based a different line of reasoning, which lends credence to the notion that SDN-enabled networks should be architected in this fashion.

Label switching can be done using MPLS [33], but MPLS support in commodity switches is limited. For example, the IBM RackSwitch G8264 only recently added MPLS support [4] and only supports 1000 MPLS push and 1000 MPLS pop rules, which we believe represents the capabilities of the underlying Broadcom Trident [10] switching ASIC as well as similar ASICs from other vendors. Thus, using MPLS to manage the core of a large network requires buying high-end switches, waiting for future switches with large MPLS tables, or both.

In this paper, we explore an alternative scalable label-switching architecture that can be implemented on existing commodity hardware by using virtual MAC addresses, which we call *shadow MACs*, as forwarding labels. Unlike traditional (physical) MAC addresses, shadow MACs are not associated with a particular physical endpoint in the network. Rather, they are opaque values akin to MPLS labels. Shadow MAC forwarding rules can be installed in the L2 forwarding tables, which are typically the largest tables available with 100,000+ entries (see Table 1). We then steer traffic on to these label-switched paths using either flexible rules at the edge (in TCAMs or vSwitches) or ARP spoofing.

In this paper, we make the following contributions:

- We demonstrate how to implement label switching with *shadow MAC* addresses as forwarding labels, which exploits large L2 forwarding tables available in commodity Ethernet hardware.
- We demonstrate that label switching solves many current SDN problems by enabling consistent network updates, scalable fine-grained forwarding, fast rerouting, and multipathing with end-to-end control.

We have implemented shadow-MAC-based switching using both OpenFlow and `expect` scripts that program our switches' CLI. The latter requires no OpenFlow support, which demonstrates that our shadow MAC mechanism can be deployed on legacy hardware.

The remainder of this paper is organized as follows. Section 2 presents the design and implementation of shadow-MAC-based label switching. Section 3 describes the benefits this approach offers. Section 4 evaluates our implementation. We compare our approach to related work in Section 5. Finally, we discuss other possible uses for shadow MACs in Section 6 and conclude in Section 7.

2. DESIGN

We propose a label-based forwarding mechanism that uses virtual Ethernet MAC addresses (*shadow MACs*) as forwarding labels. The shadow MAC mechanism can be implemented efficiently in current networks (SDN or even legacy). Shadow MACs introduce a layer of indirection in the data plane by decoupling the network core (which uses labels to forward packets) from the edge switches (which uses labels to steer packets in/out of label-switched paths). This section discusses different design components we use to implement shadow-MAC-based forwarding in commodity-switched Ethernet environments.

2.1 Control Plane

The control plane of our label-based forwarding mechanism is implemented via extensions to a centralized SDN controller. We modify the controller to export an *install route* API to install a shadow-MAC-based label-routed path to a destination. The controller then programs core and egress switches with the appropriate forwarding rules. We can activate the installed route immediately by configuring the ingress switches. Alternatively, invoking applications can pre-install multiple paths to a host and activate them later for a particular flow. This is achieved by assigning a unique identifier to each of the installed routes. SDN applications can activate one of the pre-installed routes for a flow by making an API call to the *select route* interface and specifying the source and flow identifier along with the route identifier for ingress switch match.

In practice, this model should require few changes to existing SDN applications since most already use path-oriented interfaces, e.g., Floodlight's `pushRoute()` function [15], to install rules, rather than pushing individual rules directly to switches.

2.2 Core Forwarding

The key idea in our proposal is to treat each packet's destination MAC address field as an opaque forwarding label. The SDN controller allocates a unique shadow MAC address for each path[1] in the network. It then installs rules that match on the shadow MAC address in the L2 forwarding table of each switch along the path. Switches in the network core are unaware that shadow MAC addresses do not correspond to physical endpoint MAC addresses, and simply forward packets based the installed MAC forwarding rules. This design allows switches to use their large L2-destination-based forwarding tables to implement the data plane of a label-based forwarding scheme in an scalable manner.

2.3 Edge Forwarding

Once the core is configured to forward packets based on shadow MACs, all that remains is to steer traffic in and out of MAC-label-switched paths at the source and destination edges, respectively. The source needs to select the appropriate shadow MAC based on flow information and (re)write it in each packet's destination MAC field. The destination needs to rewrite the destination MAC address from the shadow MAC to the destination's real MAC address. We have implemented two schemes to accomplish these goals, *MAC address rewriting* and *ARP spoofing*.

MAC Address Rewriting: The MAC address rewriting scheme leverages the fact that OpenFlow-compatible switches can rewrite addresses in the data plane at line rate. To steer a packet to a particular path, we install a rule in the ingress switch that matches flow-specific fields and rewrites the destination MAC address to the shadow MAC address for the desired path. At the egress switch, we install a rule that rewrites the destination MAC to the destination host's real MAC address. If end hosts are virtualized, the rewrite rules are installed in the corresponding hypervisor vSwitches. Otherwise they are installed in the physical ingress and egress switches using one TCAM rule per active shadow MAC.

The MAC rewriting scheme is very flexible—it can assign a distinct shadow MAC label to any fine-grained flow on which OpenFlow can match. Of note, this scheme allows distinct shadow MACs, and thus distinct paths through the network, to be assigned to different flows between the same source-destination host pair. Figure 1 illustrates a scenario where we use shadow MACs to configure two different routes between a source (A) and destination (B). The first route uses shadow MAC B1 to carry traffic destined to TCP port 80; the second route uses shadow MAC B2 to carry the remainder of traffic between A and B. When ingress rules overlap, as in this case, we assign higher priority to the rule that matches on more header fields.

ARP Spoofing: ARP spoofing uses the ARP protocol to steer traffic between a given source-destination host pair along a particular path. In this scheme, the SDN controller acts as an ARP proxy and handles all ARP requests from hosts[2]. When a path is activated between source and destination, the SDN controller sends a gratuitous ARP response to the source identifying the shadow MAC as the MAC address corresponding to the destination. The source host will insert the shadow MAC address in all packets intended for the destination. We then configure the destination host to accept packets addressed to the corresponding shadow MAC address either by doing rewrites (see above) or by putting the destination NIC in promiscuous mode and modifying the Linux kernel to remove MAC filtering.

[1]As described below, a path can be a linear path or a destination-rooted tree, and any given path may be used by one or more flows between source(s) and destination.

[2]Many existing SDN controllers already centralize ARP [15, 28] to provide enhanced scalability, security, and control.

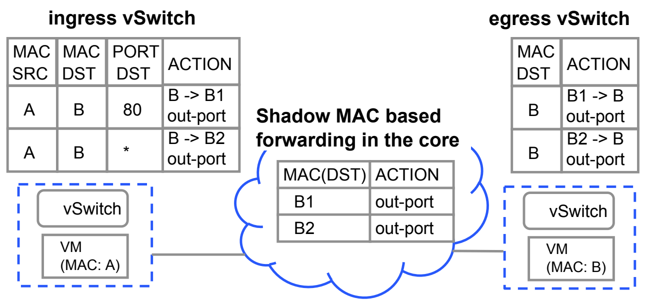

Figure 1: Address rewrite example. Two shadow MACs (*B1* and *B2*) are assigned to the destination VM with actual MAC address *B* to steer different flows from a host to different label-switched paths.

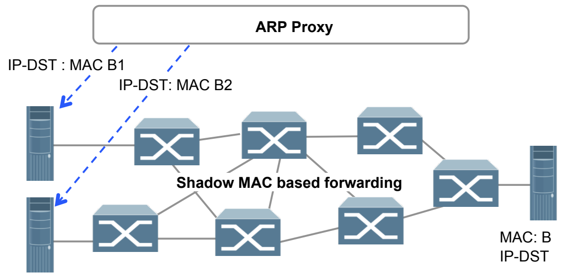

Figure 2: ARP spoofing example. Two shadow MACs (*B1* and *B2*) are assigned to the destination VM with actual MAC address *B* to steer traffic from different sources to different label-switched paths.

Figure 2 shows an example of the ARP spoofing scheme used to configure label-switched routes from two different source hosts to the same destination. The system assigns a different shadow MAC address to the same destination for the two routes. It then sends the gratuitous ARPs to the source hosts such that the two source hosts see a different shadow MAC value as the MAC address for the destination. Each host uses its shadow MAC address as the destination address.

A benefit of ARP spoofing is that it can be implemented without consuming TCAM rule space in the ingress or egress switches. A limitation of this technique is that it can only assign shadow MACs to flows at (source IP, destination IP) granularity.

3. KEY BENEFITS

Label switching using shadow MACs has several important benefits for SDN-enabled networks, including:

Minimal TCAM Usage: One benefit of our proposal is that it requires little or no scarce TCAM resources to implement fine-grained (e.g., per-flow) forwarding. Table 1, taken from [37], shows the number of TCAM and L2 entries supported by four different

Table	Broadcom Trident	HP ProVision	Intel FM6000	Mellanox SwitchX
TCAM	~2K+2K	1500	24K	0?
L2/Eth	~100K	~64K	64K	48K
ECMP	~1K	unknown	0	unknown

Table 1: 10 Gbps Ethernet Switch Table Sizes (# entries). Table reproduced from [37].

switches. The Broadcom Trident switch ASIC, used in many commercial 10 Gbps switches, contains ~25x more L2 table entries than TCAM entries. In practice, the ratio is 124x (123,904 vs. 1000) for the IBM RackSwitch G8264 [4]. This makes clear the importance of using L2 rules rather than TCAM rules when designing a network architecture intended to support fine-grained routing at scale.

As described in Section 2, we use no TCAM rules in the network core for forwarding and at most one TCAM entry per edge switch per active shadow MAC sourced or drained by a directly-connected host. In virtualized environments, MAC rewriting can be performed in the hypervisor vSwitches, reducing the physical switch TCAM requirement to zero. With ARP spoofing, no TCAM entries are used at the edge, except possibly at the egress if host stacks are unmodified. By minimizing the use of TCAM rules, our shadow MAC mechanism enables us to build large-scale fabrics with fine-grained routing, as well as leaving the TCAM free to be used for its intended purpose, e.g., ACL-based policy enforcement.

Consistent Updates: Shadow MACs can easily support per-packet consistency [32] during network updates. When the route for an ongoing flow needs to be updated, the SDN controller assigns the flow a fresh shadow MAC and installs new rules in the core and egress switches along the new path, all while ongoing traffic continues to be forwarded along the original path. When the new path is fully installed, the SDN controller updates the route atomically either by installing a new rewrite rule in the ingress switch or by sending a new gratuitous ARP. This design allows a single atomic operation to switch between old and new paths, either installing a new rewrite rule or using ARP to update the source's IP-to-MAC table. In-flight packets continue to use the old rules along the original path, while new packets use the new path and rules. Once the SDN controller is confident that all packets using the old path have drained from the network, it can tear down the old path.

End-to-End Multipathing: Many data-center topologies, e.g., fat-trees [7] and Jellyfish [35], provide multiple distinct paths between host pairs. On such topologies, it is straightforward to implement flow-based multipathing using shadow MACs. First, the SDN controller allocates multiple distinct shadow MACs and associated paths through the network for each host pair. Then, each ingress switch assigns each flow to one of the shadow MACs associated with the flow's destination. All packets in a given flow use the same shadow MAC and thus the same path to preserve packet ordering. In the extreme, each flow could be assigned its own shadow MAC. However, since there often are only a modest number of viable paths between hosts and because even the number of L2 table entries is finite, we typically install a modest number of paths between host pairs and then associate each flow with one of these paths. This scheme can be implemented efficiently in hypervisor-based virtual switches, e.g., Open vSwitch [27]. When a new flow is initiated, the vSwitch controller assigns one of the available shadow MACs to the flow. The vSwitch controller can use random, round-robin, hashing [17], or perhaps even load-aware schemes to decide which path to assign to the flow.

Though conceptually similar to ECMP [38], shadow-MAC-based multipathing provides end-to-end L2 multipathing on commodity hardware, as compared to ECMP's per-hop L3 multipathing. Further, our scheme provides more control than traditional ECMP. In ECMP, forwarding decisions are made per-hop, usually based on hashing header fields, which can result in a poor flow distribution since decisions do not exploit global information. In contrast, shadow MACs allow complete end-to-end paths to be selected.

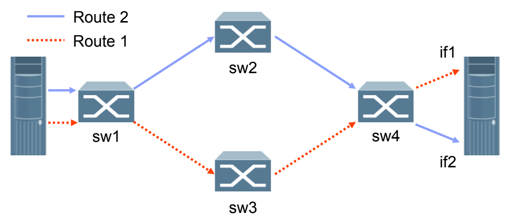

Figure 3: Diamond topology used in testbed experiments.

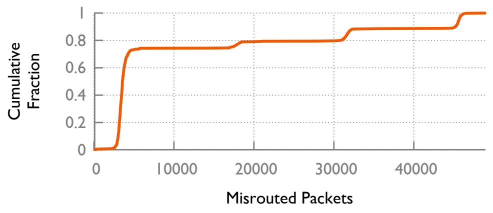

Figure 4: A CDF of the number of incorrectly routed packets when rerouting with the Static Flow Pusher in Floodlight. Our shadow MAC rerouting scheme encounters *zero* incorrectly routed packets.

Fast Switch-over: Fast switch-over to alternate routes is important for reacting to failures or rerouting flows around congestion. Our API allows SDN applications to pre-install multiple paths for a given flow, each of which uses a distinct shadow MAC address. After installation, only one is activated (by installing the appropriate ingress rewrite rule or performing the gratuitous ARP), while the remainder lie dormant until needed. Applications can initiate fast switch-over to preinstalled backup paths via another API call to activate a particular backup path. To activate a new path, the SDN controller need only replace the rewrite rule in the ingress switch or perform a gratuitous ARP, as appropriate, to cause subsequent traffic to use the new path.

4. EVALUATION

In this section, we present two experiments that illustrate the value of shadow-MAC-based forwarding. We first demonstrate the importance of consistent route updates and then compare how fast one can perform network rerouting using shadow MACs compared to traditional solutions.

Our testbed consists of four IBM RackSwitch G8264 switches running OpenFlow v1.0. We implement our shadow MAC framework in a Floodlight [15] controller module that exports a REST API to install and select routes. Our implementation consists of about 2000 lines of code.

4.1 Consistent Route Updates

To demonstrate the importance of consistent route updates, we tested how frequently packets are routed incorrectly on even a simple topology when routes are changed mid-flow. Figure 3 shows the topology and routes we evaluated.

In this experiment, a source host sends UDP packets at maximum rate via one of two routes to a destination host outfitted with one NIC per route. At the start of the experiment, the packets follow Route 1, but midway through the experiment the forwarding path is changed to use Route 2. We place destination NICs in promiscuous mode and measure the number of packets they receive.

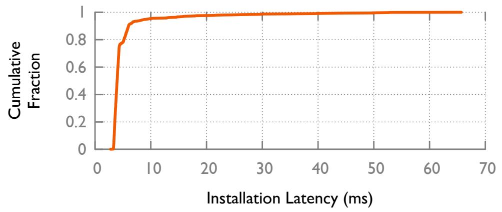

Figure 5: A CDF of the time to install a flow mod using OpenFlow barrier messages.

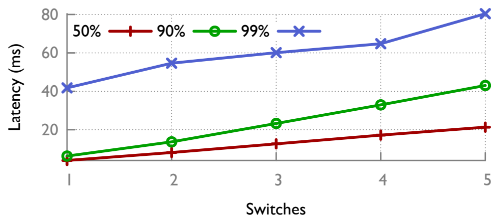

Figure 6: 50th, 90th, and 99th percentile latency (ms) to iteratively install rules on a linear chain of switches, as simulated over 1000 runs. The latency of Shadow MAC rerouting is equivalent to that of one switch.

We compare two mechanisms to change routes. Shadow MAC rerouting preinstalls rules for both routes in the non-ingress switches (sw2, sw3, sw4) and changes routes by sending a single flow mod to the ingress switch (sw1) to change the shadow MAC rewrite rule. Traditional OpenFlow rerouting uses Floodlight's Static Flow Pusher (SFP) module to install the new route by sending individual flow mods to each switch. In an attempt to minimize instability, SFP performs flow mod operations in reverse topological order (sw4, sw2, sw1). However, SFP does not implement OpenFlow barrier messages, so it is possible for rules to become active in an inconsistent order due to differences in the time it takes each switch to process the flow mod or other random events.

We consider a packet to be *incorrectly routed* if it traverses part of more than one route, e.g., a packet that traverses sw2 (Route 2) but is delivered to if1 (Route 1). We measure the number of incorrectly routed packets by comparing packet counters on the middle switches (sw2 and sw3) to the number of packets received at the corresponding destination interface (if2 and if1, respectively). If $|f|$ is the number of packets received by network element f, then in the absence of loss the number of incorrectly routed packets is: $abs(|if2| - |sw2|) = abs(|if1| - |sw3|)$.

Figure 4 plots a CDF of the number of incorrectly routed packets per run using SFP over 700 runs. At least one packet is misrouted in every run. Typically thousands of packets are misrouted because any packet queued in the network at the time of the route change will use the new route as soon as it reaches a switch with a new rule installed. In a production environment, this may cause packets to bypass a security filter or other check. In contrast, shadow MAC rerouting never misrouted a packet, because packets using Route 1 continue to use the path associated with the (original) shadow MAC. Also, shadow MAC rerouting never caused a packet to be lost, whereas SFP-based rerouting caused packet loss in ~5% of runs due to the rule for Route 2 becoming active at sw1 before it does at sw2.

4.2 Rerouting Latency

As described above, using shadow MACs allows preinstalled routes to be activated using only one flow mod. To illustrate the impact of this benefit, we measure the latency to install a flow mod in our testbed. We install flow mods using OpenFlow barrier messages and measure the latency between the `barrier_request` and `barrier_reply` at the controller. As seen in Figure 5, median latency is 3.95 ms and worst-case latency is 65 ms.

Mechanisms that install multiple flow mods sequentially will incur this latency for each hop. Mechanisms that perform path updates in parallel experience the worst latency of any individual flow mod. In contrast, Shadow MAC rerouting only requires a single flow mod. To illustrate the distinction, we simulate the latency to iteratively install flow mods along a linear path using barrier messages. In this experiment we vary path length and use the distribution from Figure 5 to simulate 1000 runs for each path length. Figure 6 presents the resulting 50^{th}, 90^{th}, and 99^{th} percentile latency. In contrast, the latency of shadow MAC rerouting is independent of the path length and equivalent to the latency of changing one switch. For 5-hop paths, shadow MAC rerouting is between 20-40 ms faster than an iterative scheme.

5. RELATED WORK

There is a significant body of work in label switching and multipathing. Moreover, in the SDN space, there have been many recent efforts to address the consistent update and switch TCAM optimization problem. In this section, we discuss some of this work.

Label-switching: Label switching is not new. MPLS [33] is the newest and most widely used form of label switching, but ATM [2], Frame Relay [3], X.25 [6], and Cisco's proprietary tag switching [5] employ similar concepts.

Casado et al. advocate label switching, and in particular MPLS, as the basis for an SDN network architecture in which an intelligent edge routes traffic in and out of a label-switched core [11]. We advocate the same architecture, but for different reasons. Casado et al. are motivated by simplicity and the ability to establish separate, clean host-network, operator-network, and packet-switch interfaces. We argue that label switching enables scalable, fine-grained forwarding and consistent network updates.

Mulipathing: Multipathing is common in modern networks, but current multipathing mechanisms like ECMP [38] and link aggregation [1, 25] make per-hop decisions. In contrast, shadow-MAC-based multipathing allows the network edge to select among end-to-end paths for each flow, even between the same two hosts. Selecting complete paths allows better optimization than local per-hop decisions.

Consistent Updates: There is a growing body of work on performing consistent updates in SDN networks [23, 32]. Our contribution is not a new model for consistent updates, but rather the demonstration that shadow-MAC-based label switching can provide per-packet consistency quite easily.

TCAM rule optimization: A large amount of work has gone into optimizing TCAM usage in SDN-enabled switches.

CacheFlow [21] proposes using the TCAM only as a cache for the most frequently used rules, while storing the full set of each switch's rules in software. DIFANE [40] partitions fine-grained rules among a number of switches and directs misses (via the data plane) to the switch that has the appropriate rule. Other work [20] focuses on calculating an optimal set of rules that respect a combination of end-point policy, routing policy, and switch rule capacity.

Inspired by PAST [37], our shadow MAC-based forwarding scheme employs non-TCAM rules in the core and a minimal number of TCAM entries in the edge, preferably installed in a vSwitch. Unlike PAST, shadow MACs support flow-granularity routing.

Finally, Metamorphosis [9] presents a design for an SDN-optimized switching ASIC. Like Intel's FM6000 [30], its design goal is to provide flexible parsers and heavily-pipelined exact-match and TCAM tables with far more rule space than existing switch ASICs. These chips would reduce the need for shadow MACs to scale SDN, but shadow MACs would further extend their system design limits.

6. DISCUSSION

This section suggests additional uses for shadow MACs and discusses how our work integrates with other trends.

Label-switched trees: While we discussed using shadow MACs to build label-switched paths, they can also be used to install label-switched, destination-rooted trees. Managing trees is more efficient than managing a collection of paths as a tree can route traffic from all sources to a destination while a path can only route traffic from one source switch. The result is essentially a version of PAST [37] where each host can be assigned multiple trees and individual flows can be assigned to distinct trees at a fine granularity.

Generic label-switched paths and trees: When installing label-switched paths (or trees), the only endpoint-specific rules are the ingress and egress rules. Thus, generic paths can be pre-installed between pairs of switches that are only made endpoint-specific when the particular ingress and egress rules are installed. We discussed one use of preinstalled backup paths in Section 2, but this idea can be extended to maintain some number of spare, pre-installed alternate paths (or trees) per switch-pair, rather per-host-pair. Doing so allows for aggressive pre-installation of alternate paths, because there are often one to two orders of magnitude fewer edge switches than hosts.

Shadow MACs vs. MPLS: Shadow MACs provide similar functionality to MPLS [33], but they do not support label stacking or label swapping as described. Without switch support for MAC-in-MAC encapsulation, there is no way to provide an equivalent of label stacking, but label swapping can be implemented using a TCAM rule to rewrite one shadow MAC with a different one.

Compatibility with overlay virtualization: The prevalence of encapsulation [12, 24, 36] in overlay virtual networks [22] dovetails well with shadow MACs. Rather than needing to rewrite MAC addresses or trick hosts into using the shadow MAC, shadow MACs simply can be used as one of a set of MAC addresses belonging to the destination hypervisor. Shadow MACs become a way to create completely generic tunnels between hypervisors.

Middleboxes: Since shadow MACs do not correspond to a physical NIC, they may confuse devices that use MAC addresses for purposes other than simple forwarding, e.g., middleboxes. There are two reasons to think that this will not cause many problems. First, we leave IP addresses untouched, so only devices that look at L2 headers are affected. Second, the previously mentioned prevalence of overlay virtual networks means that many middleboxes are being designed to strip the outer (overlay) header before processing packets. Our MAC address rewriting scheme can be implemented instead as MAC-in-MAC encapsulation and used with these newer middleboxes.

Scalability: There are two main scalability concerns: forwarding state requirements and flow setup rate.

Forwarding State Requirements: Shadow MAC forwarding has two different forwarding state requirements. State at core switches along the path and state at the ingress and egress switches. The core state is rules that fit in the large L2 forwarding tables of modern switches. This means that there are often more than 100,000 entries [18] making them an unlikely bottleneck, particularly if we use label switched trees rather than paths.

Ingress and egress forwarding state potentially requires MAC address rewriting which requires TCAM rules in most hardware switches. This is much more limited, e.g., only 1000 entries [18]. This limits how many shadow MAC paths can start or end at a given physical switch. Fortunately, in many cases, the ingress and egress switches are virtual switches which do not suffer from such stringent space limitations. Further, our ARP spoofing approach eliminates the need such rules.

Alternately, we can reduce TCAM rule space requirements at hardware ingress and egress switches by handling most traffic with normal L2 forwarding, e.g., PAST [37], TRILL [19] or plain Ethernet. Shadow MACs, and thus TCAM rule space, is then reserved for non-default routing, e.g., during route updates, for traffic engineering, and to provide backup paths for critical traffic.

Flow Setup Rate: The flow setup rate has two possible limitations. The rate at which new flows can be established in switches and the rate at which new flows can be created in the controller. Since the switches are fundamentally distributed, they should provide scale-out performance and not be a bottleneck. In the case of controllers, modern controllers can reach millions of flow operations per second on a single machine [13]. Further, the work can be instead done by a cluster of controllers providing scale-out performance. We leave this to future work.

7. CONCLUSION

We argue that combining an intelligent network edge with a label-switched core enables scalable fine-grained forwarding and consistent network updates. This design solves several pernicious problems in hardware SDN deployments, including TCAM space limitations and routing inconsistencies during network updates. We show that this architecture can be implemented efficiently on today's commodity (and even legacy) hardware by using virtual destination MAC addresses as opaque forwarding labels. Given the flexibility and advantages, we believe this approach to networking is appropriate for future SDN-enabled networks.

References

[1] 802.1AX-2008 - IEEE Standard for Local and metropolitan area networks–Link Aggregation. http://standards.ieee.org/findstds/standard/802.1AX-2008.html.

[2] Asynchronous transfer mode. http://en.wikipedia.org/wiki/Asynchronous_transfer_mode.

[3] Frame relay. http://en.wikipedia.org/wiki/Frame_Relay.

[4] IBM RackSwitch G8264 Application Guide (7.8). http://www-01.ibm.com/support/docview.wss?uid=isg3T7000650.

[5] MPLS/Tag Switching. http://docwiki.cisco.com/wiki/MPLS/Tag_Switching.

[6] X.25 : Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for terminals operating in the packet mode and connected to public data networks by dedicated circuit. http://www.itu.int/rec/T-REC-X.25-199610-I/en, 1996.

[7] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*, 2008.

[8] M. Al-fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI*, 2010.

[9] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. In *SIGCOMM*, 2013.

[10] Broadcom BCM56846 StrataXGS 10/40 GbE Switch. http://www.broadcom.com/products/features/BCM56846.php.

[11] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian. Fabric: A retrospective on evolving SDN. In *HotSDN*, 2012.

[12] B. Davie and J. Gross. A Stateless Transport Tunneling Protocol for Network Virtualization (STT). http://tools.ietf.org/html/draft-davie-stt-01, March 2012.

[13] D. Erickson. The Beacon OpenFlow Controller. In *HotSDN*, 2013.

[14] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat. Helios: A hybrid electrical/optical switch architecture for modular data centers. In *SIGCOMM*, 2010.

[15] Floodlight OpenFlow controller. http://floodlight.openflowhub.org/.

[16] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. SecondNet: A data center network virtualization architecture with bandwidth guarantees. In *CoNEXT*, 2010.

[17] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. http://tools.ietf.org/html/rfc2992.

[18] IBM BNT RackSwitch G8264. http://www.redbooks.ibm.com/abstracts/tips0815.html.

[19] J. Touch and R. Perlman. Transparent Interconnection of Lots of Links (TRILL): Problem and Applicability Statement. RFC 5556, May 2009.

[20] N. Kang, Z. Liu, J. Rexford, and D. Walker. Optimizing the 'One Big Switch' abstraction in Software Defined Networks. In *CoNEXT*, 2013.

[21] N. Katta, J. Rexford, and D. Walker. Infinite cacheflow in sofware-defined networks. Technical Report TR-966-13, Princeton Computer Science, 2013.

[22] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, N. Gude, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet, S.-H. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, R. Zhang, and VMware. Network virtualization in multi-tenant datacenters. In *NSDI*, 2014.

[23] R. Mahajan and R. Wattenhofer. On consistent updates in software-defined networks. In *HotNets*, 2013.

[24] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-08, February 2014.

[25] MC-LAG. http://en.wikipedia.org/wiki/MC_LAG.

[26] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. PortLand: A scalable fault-tolerant layer 2 data center network fabric. In *SIGCOMM*, 2009.

[27] Open vSwitch. http://www.http://openvswitch.org.

[28] OpenDaylight. http://www.opendaylight.org/.

[29] OpenFlow-switch. https://www.opennetworking.org/standards/openflow-switch.

[30] R. Ozdag. Intel Ethernet Switch FM6000 Series - Software Defined Networking. http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ethernet-switch-fm6000-sdn-paper.pdf.

[31] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca. Planck: Milisecond-scale monitoring and control for commodity networks. In *SIGCOMM*, 2014.

[32] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In *SIGCOMM*, 2012.

[33] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. http://tools.ietf.org/html/rfc3031.

[34] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore. OFLOPS: An open framework for openflow switch evaluation. In *PAM*, 2012.

[35] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey. Jellyfish: Networking data centers randomly. In *NSDI*, 2012.

[36] M. Sridharan, A. Greenberg, Y. Wang, P. Garg, N. Venkataramiah, K. Duda, I. Ganga, G. Lin, M. Pearson, P. Thaler, and C. T. and. NVGRE: Network Virtualization using Generic Routing Encapsulation. http://tools.ietf.org/html/draft-sridharan-virtualization-nvgre-04, February 2014.

[37] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter. PAST: Scalable ethernet for data centers. In *CoNEXT*, 2012.

[38] D. Thaler and C. Hopps. Multipath issues in unicast and multicast next-hop selection. http://tools.ietf.org/html/rfc2991.

[39] R. Wang, D. Butnariu, and J. Rexford. OpenFlow-based server load balancing gone wild. In *HotICE*, 2011.

[40] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable flow-based networking with DIFANE. In *SIGCOMM*, 2010.

[41] T. Yuzawa. OpenFlow 1.0 Actual Use-Case: RTBH of DDoS Traffic While Keeping the Target Online. http://packetpushers.net/openflow-1-0-actual-use-case-rtbh-of-ddos-traffic-while-keeping-the-target-online/, April 2013.