Using MAC Addresses as Efficient Routing Labels in Data Centers

Arne Schwabe
University of Paderborn
Warburger Straße 100
33098 Paderborn, Germany
arne.schwabe@uni-paderborn.de

Holger Karl
University of Paderborn
Warburger Straße 100
33098 Paderborn, Germany
holger.karl@uni-paderborn.de

ABSTRACT

A number of new technologies such as cloud services and/or virtualization have changed data center networks in the last few years. The benefits of the techniques are clear but the downside is that more forwarding entries are needed in network switches to support these techniques. Unfortunately, the number of forwarding entries in switches have a hard limit.

We give a formal problem definition for minimizing the number of forwarding entries and a proof that the problem is $\mathcal{NP}$ complete.

We show that the destination MAC address can be used as a universal label in software-defined networks and the ARP caches of hosts can exploited as an ingress label table, reducing the size of the forwarding tables of network devices. We have the additional advantage of not requiring a special type of data center network or additional hardware capabilities.

We demonstrate that our technique can solve the problem of FIB sizes by introducing a greedy scheme for all pairs ECMP with a minimal number of FIB entries.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Network Architecture and Design

Keywords

Software Defined Networking; Data Plane; OpenFlow;Label routing

1. INTRODUCTION

In a modern switch, forwarding is implemented in hardware to forward at line rate. A key component of forwarding is the lookup table which is implemented by using Ternary Content-Addressable Memorys (TCAMs). TCAMs are expensive in energy and hardware cost. This limits the number of forwarding entries a switch can store in hardware. Forwarding in a data center is the lookup of the destination MAC address. MAC addresses of hosts are usually unstructured and each host needs an entry in the forwarding table. Reducing the number of entries will save costs and energy.

Usually, the number of required entries is reduced either by using compression/combining multiple entries or by storing only a subset of the entries in hardware tables. Combining multiple entries is difficult and even good approaches only achieve compression rates of 50% [10]. The other approach, to store only a subset of the entries in the hardware tables, introduces an additional delay whenever a packet arrives which has no corresponding entry in the hardware table and an entry has to be fetched, e.g., from a central controller as in SDN or locally computed. Both solutions only alleviate the problem and provide no real solution. For these reasons the commercial switches currently have to provide a very large number of possible entries.

The root of the problem is the lookup of unstructured addresses. One solution is to avoid the lookup of MAC addresses and instead perform the lookup on a label which can be controlled and given a structure. The idea of using labels to improve or enable forwarding hardware is quite old, dating back at least to ATM and MPLS. To benefit most from using labels for forwarding, labels should be added to the packets as early as possible and removed as late as possible. Typically, the ingress switch adds the label to a packet and the egress switch strips the labels from the packets. This allows all switches between the ingress and egress switch to benefit from doing label lookups instead of MAC address lookups.

The label can be added in two ways. The first way is to encapsulate the packet, e.g., adding an MPLS header, on the ingress switch and decapsulate the packet on the egress switch. The second way is to replace the content of a header field with the label at the ingress switch and, if necessary, reverse the replacement at the egress switch. Approaches specially designed for the problem in the data center like Portland [8] use the second approach and replace the source and destination MAC addresses of the packets.

Using labels for forwarding shrinks the forwarding table when structured labels are used. Structured labels enable assigning similar labels to packets forwarded in the same way and thus one forward entry can be used to match multiple labels. Adding/removing labels needs multiple entries at the ingress and egress switches. The work of adding addresses to the packets is even duplicated at the hosts and the ingress/egress switches. The hosts add source and destination MAC addresses according to their own addresses

HotSDN'14, August 22, 2014, Chicago, Illinois, USA.

ACM 978-1-4503-2989-7/14/08 ...$15.00.
http://dx.doi.org/10.1145/2620728.2620730.

and ARP tables; the switches will then replace the addresses according to their tables.

Modifying the hosts (respectively, their operating system) to directly write the right MAC addresses (containing the labels) is the apparent solution but modifying hosts is often unacceptable. Without modifying the hosts, the source MAC addresses are always the hosts' own addresses which cannot be controlled. But the ARP table of the hosts is filled from ARP replies and this can indeed be controlled. The question arises how far this limited control over the hosts can be exploited to eliminate the duplicate work and further improve the gain from using labels for forwarding. We will show in this paper that this is indeed feasible and desirable.

In the remainder of the paper, we will first, in Section 2, compare the different approaches and how these approaches deal with problem and then, briefly in Section 3, recap switch lookup capabilities for forwarding packets. Based on these capabilities we will define in Section 4 a model which describes the minimal number of forwarding entries for a given network and then show the $\mathcal{NP}$-completeness of the problem in Section 5. In Section 6 we will show that rewriting the MAC addresses at ingress switch is not needed and that we can use MAC addresses as labels with a greater flexibility than other approaches. To solve the problem we present an ILP formulation of the problem and also provide a greedy heuristic. In the last two sections we will evaluate the algorithm and give a conclusion.

2. RELATED WORK

Reducing the size of the forwarding information base (FIB) has been studied for various different scenarios and requirements. A very general approach is to use compact routing algorithms designed for arbitrary networks. For example, the compact routing scheme of [11] has a table size of $\tilde{\mathcal{O}}(n^{1/2})$ with a stretch factor of 3.

One very general proposal aiming to reduce the FIB table size is Pathlet routing [4]. Pathlet routing sets out to reduce the FIB size/complexity of inter-domain routing while retaining the flexibility of the policy routing possible when using BGP. Pathlet routing achieves this flexibility by adding labels to the packets that encode parts of the path and allows each AS to decide trade-offs between the number of FIB entries and complexity of the implemented routing policy.

The question arises if FIB reduction techniques aimed at inter-AS communication [12], [3] can be applied to the data center FIB reduction discussed here. Common to both problems is that the identifier used for forwardeding, MAC addresses in the data center and IP prefixes in the inter-AS communication are unstructured. Directly translating these proposals does not work well because it makes switches to ASes and the directly connected MAC addresses to the networks of the AS. Carefully adapting the proposals to data center networks is not straightforward and thus creates new protocols.

Instead of matching the header directly with TCAMs, the approach in [9] uses Bloom filters for matching addresses. Bloom filters can have false positives. The approach avoids false positives by requiring a large number of alternative paths between communication pairs and implementing a strategy that avoids forwarding over paths which trigger false positives. The requirement of a special type of network and the modification of hardware to support Bloom filters make this approach only viable in some scenarios.

PortLand [8] assigns each switch a hierarchical level of either core, aggregation or edge. A part of the network consisting of aggregation switches and edge switches are called a "pod". For every host, the pod and edge switch are encoded into a "pseudo mac" address. Portland uses the pseudo MAC address for forwarding packets on the aggregation and core switches. Ingress switches rewrite the source MAC addresses to the pseudo MAC address. The egress switches replaces the pseudo MAC with the real address of the host again. PortLand maintains a one-to-one mapping between pseudo MAC addresses and hosts. The strict encoding of host positions into the MAC address also limits PortLand to topologies matching PortLand's network model.

While PortLand maps one IP address to exactly one MAC address, first hop redundancy protocols (HSRP[6] or VRRP[7]) let multiple IP addresses share a single MAC address for receiving packets on the failover IP gateway address but their own MAC address for forwarding packets to the hosts. Using this "virtual" MAC address has the advantage that packets with this MAC address will always be accepted by the routers and the ARP table of the client, always has a valid entry as long as one of the routers is still working. When forwarding a packet to a client a router will use its own MAC address.

3. BACKGROUND

This background section recaps the important aspects of lookup tables in switch hardware.

3.1 Lookup hardware

In a switch an incoming packet is matched against the FIB of the switch. This is done by looking up multiple header fields from the packet itself as well as using meta information including the ingress port of the packet. The lookup is either made as an exact match, e.g., an IP address equaling 1.2.3.4, or as a wildcard match such as an IP address matching 5.6.0.0/16, often combined with a priority to give longer prefix entries a higher priority.

This matching is implemented using TCAMs and allows fields to match against 0, 1 and "don't care", usually written as X. As an example the wildcard 11100XXX XXXXX1100 matches all 16-bit words beginning with 11100 and ending with 110.

Wildcard matches can be seen as an indicator function for bit strings of length n:

$$t : \{0,1\}^n \mapsto \{0,1\}, \quad t(x) = \begin{cases} 0 & \text{wildcard matches } x \\ 1 & \text{otherwise} \end{cases}$$

We define T_n to be the set of all possible wildcard functions with n bits input size.

3.2 Forward Information Base

Regardless of the routing algorithm/forwarding strategy used in the network, a FIB entry consists of a matching and a forward action.

To replace two FIB entries with one entry, two conditions have to be met. First, the matching of the new rule must match everything the old two rules matched and not match anything the old rules did not match. Second, the forwarding actions of both entries have to be the same. If the output actions differ, combining two entries will change the semantic. Merging multiple FIB entries is often impossible since the

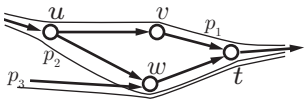

Figure 1: Small network with three paths (thin lines)

inputs (like the MAC addresses of hosts) have no structure that allows grouping them together in a wildcard.

4. MODEL

To analyze and solve the problem of minimizing the forward table size we need a formal definition of the problem which simplifies the problem without restricting it.

When a packet is forwarded in a network, the packet is forwarded to an outgoing port on each switch on its path until the packet arrives at its destination. We use the paths that the packets should be forwarded over and the graph itself as the input for our problem.

For Ethernet link layer forwarding the forward action is "output packet on port x" and fits perfectly into the model. A more complex routing algorithm will install different forward actions with the same egress port. This can be represented in our model as two different outgoing edges going to the same switch.

When constructing the wildcards to match the labels only packets passing a switch need to be considered. For example, in Figure 1 a wildcard for the edge (u, v) must match the label of p_1 and must not match the label of p_2, but it is irrelevant if the wildcard matches p_3.

Path label assignment.

Let n be the number of bits used for the label. Given a graph $G = (V, E)$ and loop-free paths $P \subset \mathcal{P}(E)$. Does a mapping $m : P \mapsto \{0,1\}^n$ exists so that for each edge $e = (u, v)$ a function $t_e \in T_n$ exists with $t_e(m(p)) = 1$ if $e \in p$ and $t(m(p)) = 0$ if $(u, w) \in p$ with $w \neq v$ for all paths $p \in P$.

5. COMPLEXITY OF THE PROBLEM

The path label assignment problem is $\mathcal{NP}$-complete for inputs of arbitrary networks. This section will give a proof of this.

The existence of a polynomial-sized ILP program (see Section 7.1) for the problem shows that the problem is inside $\mathcal{NP}$ since ILP problems are solvable with an $\mathcal{NP}$ algorithm. To establish $\mathcal{NP}$-completeness we will show that the 4-colorability problem [1, 2] (can four colors be assigned to vertices so that no edges connect vertices with the same color) can be reduced to the path label assignment problem.

Let $G = (V, E)$ be the input for the 4-colorability problem. For each edge $e_i \in E$ we add a switch r in our model with two output ports. Each vertex $v_j \in V$ is identified with a path p_j. For every edge e_i in the graph, we add a router r_i with two output ports and assign to each one path corresponding to one of the two vertices of the edge.

By choosing n, the number of label bits, as two, there are four possible label values for each path (00, 01, 10 and 11). Figure 2 shows the idea of the reduction. For a switch with one path per output port each path must have a different bit mask to be distinguishable.

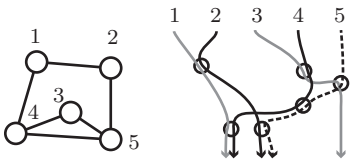

Figure 2: Example transformation of a graph for 4-colorability

If the path label assignment problem has a solution, set the colors of the vertices v_i in G according to the bits of the corresponding paths p_i to solve the 4-colorability problem. To show that this is indeed a coloring solution, assume that this is not a valid solution for the 4-colorability problem. Then an edge $e = (u, v)$ exists which connects two nodes with the same color. The paths corresponding to u and v, p_u and p_v, have the same bit mask. Since p_u and p_v are distinguishable in r_e they cannot have the same bit mask.

If the label assignment problem has no solution the 4-colorability problem also has no solution. Again, assume the bit mask problem has no solution but the 4-color problem has a solution. Assign each color a bit mask and the bit masks to the paths in the bit mask problem. Then, for each switch the paths will have different bit masks and the bit mask problem has a solution, establishing the contradiction.

6. MAC ADDRESSES AS LABELS

Using software-defined network (SDN) gives a much greater control over the network. We use this greater freedom to repurpose the destination MAC address as a flexible forwarding label. This use of the MAC address has the big advantage that labeling packets can be offloaded to the host by the ARP protocol rather than requiring a FIB entry to add the label on every ingress switch.

Hosts on the network rely on ARP/NDP to learn the MAC address of other devices. Instead of delivering the ARP query/neighbor discovery packets to the hosts, an SDN can intercept these packets. This allows us to respond with arbitrary MAC addresses. Intercepting and modifying the ARP request instead of attaching a separate label to the packets has an important advantage: the host will put the received MAC address into its own ARP cache and will put the MAC destination address into all outgoing packets to that particular IP address. This removes the need to label the packets on the ingress switch. Effectively we use the ARP table of the hosts to store the entries we otherwise need to store in the FIB tables of the ingress switches.

Answering the ARP queries allows us to answer with a label for the path to the destination IP address. If a different host uses ARP to query the same IP we can respond with a different MAC address. Effectively we have the possibility not only to use structured MAC addresses for hosts but even individually for each source and destination IP pair without requiring additional FIB entries at the ingress and egress switches.

When a packet arrives at the egress, the packet still carries the label as destination MAC address. Without modifying the destination host operating system, the host will drop the packet since the destination MAC address is not matching its own MAC address. Hence, the switch needs to replace the destination MAC with the real MAC address. Figure 3

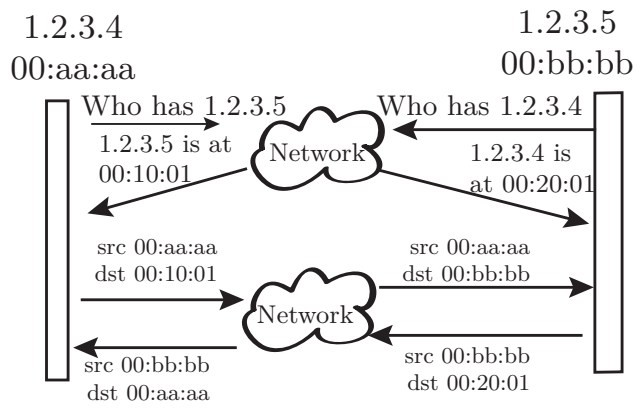

Figure 3: Intercepting and modifying ARP packets in a SDN network

shows the resulting packet flows. Having to do this extra step to undo the labeling seems to contradict the idea of using ARP to label packets. But the important difference is that labeling has to be on every ingress switch, while rewriting the MAC address needs only to be done on the egress switch. Since the egress switch needs a FIB entry to forward the packets to the port in any case, this only adds an additional action to the already existing entry.

Using the destination MAC address as a forward label clearly breaks the assumption that the source and destination address of a host are always the same for the Ethernet layer. The ARP table of the receiving host contains a label MAC address for the source IP address instead of the real MAC address of the host. Our approach does not modify the source MAC address, which is the physical address of the sending host, to avoid adding FIB entries in the ingress switch. For a received packet the source MAC address will differ from the address stored in the ARP table. Nevertheless, the host will accept the packet; the first hop redundancy protocols are working in a similar way, albeit in a much more limited scope, and we do not violate the standards for Ethernet end devices.

7. SOLVING THE PROBLEM

In this section we present two methods for solving the problem. We first show an exact solution with a integer linear program (ILP) and a greedy heuristic.

7.1 ILP solution

In this section we model the problem as an ILP. The variables of the ILP are defined as followed:

t_{ej}	value of bit j of wildcard t_e
x_{ej}	bit j of wildcard t_e is a "don't care"
p_{ij}	value of jth bit of the label for path i
n_{ejk}	bit k of path j label is not matched by t_e
d_{ijk}	decision variable for n_{ijk}

We model the wildcard functions t_e by using t_e and x_e. The mapping of the path to an n-bit label is modeled by the p_{ij} variables. All variables are binary. n_{ijk} can be changed to an arbitrary float without changing the solution of the ILP since the constraints will force the variables to be either 0 or 1.

Further, we define for an edge $e = (u, v)$ the function $s(p_j, e)$ to be 1 iff p_j includes an edge (u, w) with $w \neq v$.

$$x_{ek} \geq p_{jk} - t_{ik} \tag{1}$$
$$x_{ek} \geq t_{ek} - p_{jk} \tag{2}$$

$$k = 1 \ldots n,\ \forall e \forall p_j :\ e \in p_j$$

$$\sum_{k=1}^{b} x_{ek} \geq 1 \qquad \forall e \tag{3}$$
$$n_{ejk} \leq t_{ek} - p_{jk} + (1 - d_{ejk}) \cdot M \tag{4}$$
$$n_{ejk} \leq p_{jk} - t_{ek} + d_{ejk} \cdot M \tag{5}$$
$$n_{ejk} \leq 1 - x_{ek} \tag{6}$$
$$\sum_{k=1}^{n} n_{ejk} \geq 1 \tag{7}$$

$$k = 1 \ldots n,\ \forall e \forall p_j : s(p_j, e) = 1$$

The first block of constraints (1-3) makes sure that a wildcard for an edge matches all labels of paths containing that edge ($e \in p_j$). Constraints 1 and 2 ensure that t_{ek} and p_{jk} (bit k of wildcard and path label) are the same if x_{ik} is 0 (not a "don't care"). Constraint 3 ensures that every wildcard has a least one bit that is not set to "don't care".

To ensure that the wildcard only matches labels it should match, Constraint 7 ensures that at least one bit of a label exists that is not matched. Constraints 4 and 5 ensure that n_{ijk} can be only 1 if p_{jk} and t_{ek} have different values. Constraint 6 furthermore ensures that n_{ijk} is 0 if the kth is a "don't care".

The ILP has no optimization goal since the problem is either solvable or not.

7.2 Greedy Heuristic

Unfortunately, calculating an optimal solution using the ILP does not yield a solution for any problem instances in a reasonable time except for very small ones (less than a 10 vertices) . To implement the idea in a real-world scenario, a faster algorithm is needed.

The Ethernet MAC address has no variable length but a fixed number of 48 bits. Laying 16 bit aside to differentiate multiple (virtual) hosts behind a single switch port gives a usable amount for the label of 32 bit. An approximation algorithm should gracefully adapt to a situation where a perfect solution requiring the only n-bit wildcards is not possible.

To achieve this goal we designed a greedy algorithm. The idea is to set one bit after another for every edge in a way that brings the solution closer to requiring only one wildcard per link. For each bit we will consider the switches in a random order and assign the bit values to the path in a locally optimal way.

The algorithm works as follows: uniformly at random select an edge e. Determine the bits that all paths that include the edge have in common. Use these bits as a wildcard on all other edges of the same switch. Put any path that matches the wildcard in the set U. This set U now contains the paths that cannot be distinguished from e with the wildcard. If the set of U is empty, move to the next edge. Otherwise,

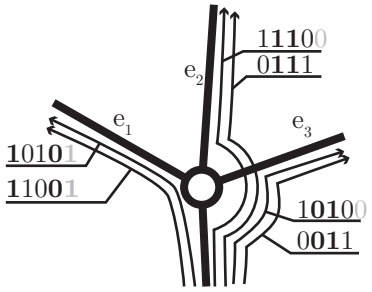

Figure 4: Node with six paths and four already assigned bits, common bits in bold, next bits in gray

set the unset bit of the paths in U so that the set U is minimized. Continue with the algorithm until either the set of indistinguishable edges is empty for every edge or when the number of bits is reached.

As an example, consider Figure 4 where the first four bits are already set. A wildcard using the common bits of all paths including edge e_1 is 1XX0 which matches one path of e_2 and one of e_3. Adding 0 to the paths of e_1 and 1 to the paths of e_2 and e_3 makes the wildcard 1XX01 only match only paths of e_1.

After the bits have been set for every path and U is empty for every edge only one wildcard is needed per edge. For the infeasible case, we use a greedy second phase of the algorithm to find a valid set of wildcards. Since using bits that are common to all paths do not create an empty set U we split the wildcard into two wildcards w_0 and w_1. We calculate the sets U_1 and U_0 for all bits which are not common between all paths. Then we choose the bit which minimizes U_1 and U_0. We repeat this step until the sets U_i are empty for all wildcards w_i of the path.

8. EVALUATION

Our evaluation consists of two parts. The first part shows that our method of using the destination MAC address as labels without rewriting the source address (Section 6) works as anticipated. The second part is an empiric evaluation of the greedy heuristic in multiple network scenarios.

While our methods comply with the standards, the behavior of the network is unusual from the operating system's perspective. We built a test bed using Mininet [5] and connected various virtual and physical hosts with different operating systems (Windows, Linux, Mac OS X and Cisco IOS) to it and implemented our approach on the test bed.

Our findings confirmed that the operating systems will accept IP packets for their own IP address as long as the destination MAC address is right. The source MAC address can be arbitrary. Or from an Ethernet layer-centric view, the operating systems do not make assumptions about the network addresses other than the receiving MAC address should be its own address.

The second part is a simulation to evaluate the possibility of reducing the number of needed flow table entries by using the greedy algorithm described in Section 7.2.

For small or simple structured networks (for example, trees with less than 100 switches) the greedy heuristic achieves the optimal solution with one wildcard per edge.

As a more challenging example for the greedy heuristic we built a CLOS network consisting of 2 core switches, 16 pairs of distribution switches (2 up links each). Each distribution pair switch had 8 top of rack switches connected to it. The network has 320 links between the switches (or 640 as unidirected links). To test the robustness of the heuristic we modified the graph by randomly removing links and switches. As paths we calculated all shortest paths between all switches. Using these paths the SDN controller can choose the exact path between two end hosts purely by answering an ARP reply and without having to install or modify any FIB entries. The resulting number of wildcards compared to edges in the network is plotted in Figure 5. In this complex setup the heuristic manages to achieve an average of about 4-5 wildcards per outgoing edge. The number of required wildcards is quite stable for the modified graphs as well.

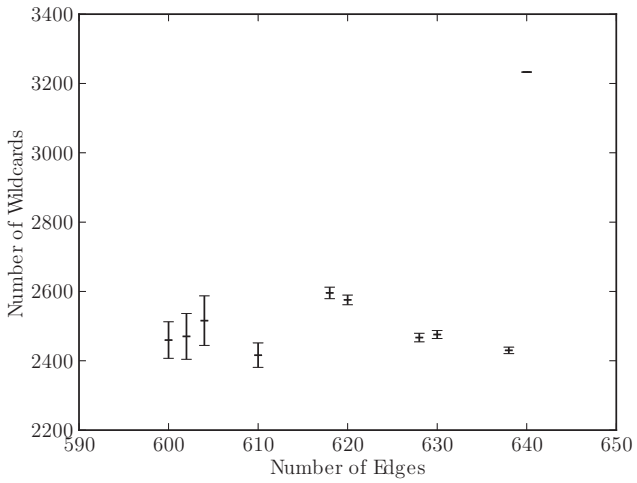

Figure 5: Average number of wildcards used by the greedy heuristic with 95% confidence intervals

9. CONCLUSION

We have formalized the problem of finding a optimal number of FIB entries for a network and proofed the complexity of the problem.

We have shown that our techniques for reducing the number of needed flow table entries in a software defined network are viable. A centrally managed network makes it possible to use the destination MAC address as very lightweight label that is applied for free by the connected hosts allowing very small FIB tables and label routing. It also allows us to match the limits defined by our formal definition. We have provided a greedy algorithm which can be used to calculate labels for arbitrary networks.

Acknowledgment

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901).

10. REFERENCES

[1] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[2] D. P. Dailey. Uniqueness of colorability and colorability of planar 4-regular graphs are np-complete. *Discrete Mathematics*, 30(3):289 – 293, 1980.

[3] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. The Locator/ID Separation Protocol (LISP). RFC 6830 (Experimental), Jan. 2013.

[4] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica. Pathlet routing. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, pages 111–122, New York, NY, USA, 2009. ACM.

[5] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

[6] T. Li, B. Cole, P. Morton, and D. Li. Cisco Hot Standby Router Protocol (HSRP). RFC 2281 (Informational), Mar. 1998.

[7] S. Nadas. Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6. RFC 5798 (Proposed Standard), Mar. 2010.

[8] R. Niranjan 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. *SIGCOMM Comput. Commun. Rev.*, 39(4):39–50, Aug. 2009.

[9] C. E. Rothenberg, C. Macapuna, F. Verdi, M. Magalhães, and A. Zahemszky. Data center networking with in-packet bloom filters. In *Proc. SBRC*, pages 553–566, 2010.

[10] O. Rottenstreich, M. Radan, Y. Cassuto, I. Keslassy, C. Arad, T. Mizrahi, Y. Revah, and A. Hassidim. Compressing forwarding tables. In *IEEE Infocom*, 2013.

[11] M. Thorup and U. Zwick. Compact routing schemes. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '01, pages 1–10, New York, NY, USA, 2001. ACM.

[12] X. Yang, D. Clark, and A. W. Berger. Nira: A new inter-domain routing architecture. *IEEE/ACM Trans. Netw.*, 15(4):775–788, Aug. 2007.