# Using Mac Addresses as Efficient Routing Labels in Data Centers

Arne Schwabe
University of Paderborn
Email: arne.schwabe@uni-paderborn.de

Holger Karl
University of Paderborn
Email: holger.karl@uni-paderborn.de

***Abstract*—With advent of software defined networks (SDNs) the centrally coordinated routing paradim has been in the focus. With central routing fine grained traffic engineering and flow methods is pushed to environments like data centers that traditionally did not make heavy use of these techniques.**

**The benefits of the techniques are clear but the downside is that more forward entries are needed to support these techniques. Unfortunately the number of forwarding entries in switches have a hard limit.**

**We show that the destination MAC address can be used as universal label in software defined networks and the ARP caches of hosts can exploited as ingress label table and therefore reduce 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.**

## I. Introduction

The central element in a switch is the forwarding information base (FIB) which holds the entries to match and forward the packets.

To reduce the number of FIB lookup misses often FIB entries are installed which not only match the flow causing the FIB miss but also match similar flows, which e.g. have the same source and destination IP but different TCP/UDP ports. Similar flows will then matched by the same rule and do not experience the delay for the first packet of the flow. In order to achieve optimal performance, FIB lookup misses can be completely avoided if entries for all possible flows are present in the FIB. This naturally only works if FIB hold all required entries.

Additional requirements and policies for routing in the network as quality of service and network engineering will result in additional FIB entries. Trying to minimize the number of required FIB entries to be able to accommodate as many requirements as possible is the natural consequence. FIB lookup tables are implemented using Ternary Content-Addressable Memorys (TCAMs), which are expensive in cost and power. Reducing the number of FIB entries also reduces the number of entries in the TCAMs, which can result in power/cost saving. [8], [13]

Following we will present an approach that will work using existing network hardware capabilities. We first give a model of a current hardware FIB. We define the problem of minimizing FIB entries for data centers and analyze the complexity of the problem. In Section 3 we explain the technique of using MAC addresses as label. We provide a heuristic algorithm to solve the problem in a practical usable time.

## II. Model

In this paper we will refer to any device connected to an SDN-enabled switch as a “host” and to the

SDN-enabled switches simply as "switch". We further assume that all switches in the data center are SDN enabled switches.

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 like ingress port of the packet. The lookup is either made as an exact match, like IP address equaling 1.2.3.4, or as a wildcard match like 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 $X$. For example the wildcard 11100XXX XXXXX1100 matches all 16 bit words beginning with 11100 and ending with 110. Wildcard matches can be seen as a 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.

Regardless of the routing algorithm/forwarding strategy used in the network, a FIB entry consists of a matching rule and a forward action. To replace two FIB entries with one entry two condition 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 action of both entries has to be the same. If the output action differ, combining two entries will change the semantic. Merging multiple FIB entries is often impossible since the inputs like the MAC addresses of hosts have no structure that allows grouping them together in a wildcard.

For a Ethernet link layer forwarding the forward action is as simple as "output packet on port x" and matching is done by looking at the destination MAC address. More complex routing algorithm will install different forward actions with the same outport port. For the context of merging forward entries these different forward actions are like two different output ports.

A solution is to use inputs that make merging easier is the use of labels for routing. The disadvantage of using labels for routing is that usually the ingress switch needs to label the incoming packets, which translates in additional FIB entries. To efficiently use labels we need to solve two problems: The first one is finding labels themselves and the second is using the labels without increasing the number of FIB entries significantly.

A MAC address (or any other label) is just an $n$ bit long string. A communication between two hosts is expressed as a path from one node to another in the graph. Multiple actions are present for one port this can be represent as a multiple edges between vertices as previously stated. This allows us to simply all routing decision to the decision of which edges a path contains. We can formulate of using minimal number of FIB entries:

### A. Path label assignment

Let $n$ be the number of bits used for the label. Given a graph $G = (V, E)$ and number of loop-free paths $P \subset \mathcal{P}(E)$. Does a mapping $m : P \mapsto \{0,1\}^n$ exist, 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 $(u, w) \in e$ with $w \neq v$.

### B. Complexity of the problem

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

The existence of a polynomial-sized ILP program (see Section 2.3) for the problem shows that the problem is inside $\mathcal{NP}$ since MILP 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 connects 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 by a path $p_j$. For every edge $e_i$ in the graph we add an 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 the number of bits ($n$) as two there are four possible label values for each path (00, 01, 10 and 11). Figure 1 shows the idea of the reduction. For a switch with one path per output port each path must have a different bitmask to be differentiatiable.

If the path label assignment problem has a solution, set to colors of the vertices $v_i$ in $G$ to according the bits of the corresponding paths $p_i$ to solve the 4-colorability problem . Assume 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. Assume the bit mask problem has no solution but the 4-color problem has 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.

### C. Exact MILP solution

In this section we model the problem as a mixed-integer linear program (MILP).


Fig. 1. Example transformation of a graph for 4-colorability

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 $j$th bit of path $i$ label |
| $n_{ejk}$ | bit $k$ of path $j$ label is not matched by $t_e$ |
| $d_{ijk}$ | decision variable for $n_{ijk}$ |

We model the three valued functions $t_e$ by using $t_{ij}$ and $x_{ij}$. The mapping of the path to an $n$-bit string is modeled by the $p_{ij}$ variables. All variables are binary. $n_{ijk}$ can be changed to a arbitrary float without changing the solution of the ILP since the equations will force these 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 $w \neq v$.

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

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

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

$$\sum_{k=1}^{b} x_{ek} \geq 1 \qquad \forall e \tag{3}$$

$$n_{ejk} \leq t_{ek} - p_{jk} + (1 - d_{ejk}) \cdot M \tag{4}$$

$$n_{ejk} \leq p_{jk} - t_{ek} + d_{ejk} \cdot M \tag{5}$$

$$n_{ejk} \leq 1 - x_{ek} \tag{6}$$

$$\sum_{k=1}^{n} n_{ejk} \geq 1 \tag{7}$$

$$k = 1 \ldots n,\ \forall e \forall p_j : s(p_j, e) = 1 \tag{8}$$

The first block of equations (1 - 3) makes sure that a wildcard for an edge matches all labels of paths containing that edge. ($e \in p_j$) Equation 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 an "don't care"). Equation 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 Equation 7 ensures that at least one bit of a label exists that is not matched. Equations 4 and 5 ensure that $n_{ijk}$ can be only 1 if $p_{jk}$ and $t_{ik}$ have different values. Equation furthermore ensures that $n_{ijk}$ is 0 if the $k$th is a "don't care".

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

## III. MAC ADDRESSES AS LABELS

Using an SDN gives a much greater control over the network. We use this greater control to not rely on backward learning. This allows use to use the destination MAC address as generic forward 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 an FIB entry to add the label on every ingress switch.

In a normal Ethernet network each client uses the same MAC address for receiving and sending packets. This allows the switches to learn which destination MAC addresses are mapped to each port by observing the packet's source MAC addresses. Hosts on the network rely on higher-level protocols (ARP/NDP [12], [10]). Figure 3 shows an example of traditional ARP query/reply communication.

Instead of delivering the ARP query/neighbor discovery packets to the hosts, the network can intercept the packets. This allows us to respond with arbitrary MAC addresses. Doing so allows us to answer with a label that for that path from the host to destination IP address. This breaks the assumption that source and destination address of a host are always the same for the Ethernet layer.

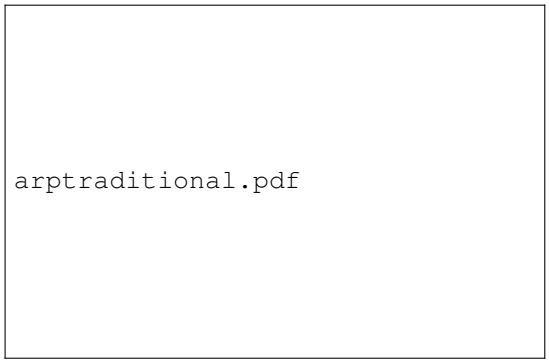


Fig. 2. ARP resolution and resulting communication in a traditional layer 2 network


Fig. 3. Intercepting and modifying ARP packets in a SDN network

Intercepting and modifying the ARP request instead of attaching a separate label to the packets has an important advantage. The host will put that MAC address into its own ARP cache and will already put the MAC destination address into all outgoing packet for that particular IP address. That removes the need to label the packets on the ingress switch in the network to be able forward packets according to a label. Effectively we use the ARP table of the hosts to store some of the entries we otherwise would need to store in the FIB tables of the switches.

When packet arrives at the egress the switch the packet still carries the label as destination MAC addresses. Without modifying the destination host oper-


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

ating system, the host will drop the packet since the destination MAC address is not matching its own MAC address. The switch needs to replace the destination MAC with the real MAC address. Figure 3 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 be done on the egress switch. Since the egress switch needs a FIB entry to forward the packets to the port, this only adds an additional action to the already existing entry.

The ARP table of the receiving host contains a label MAC address for source IP address instead of the real address of the host. Our approach does not modify the source MAC address, which is the physical address of the sending host. For a received packet the source mac address will differ from the address the address store in the ARP table. This raises the question if the host accepts these packets.

## IV. Assigning labels

Unfortunately, calculating an optimal solution using the MILP described presented in Section 2.3, does not yield a solution for any but very small problems instances in a reasonable time. 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 $8$ bit aside to differentiate multiple (virtual) hosts behind a single switch port gives a usable amount for the label of 40 bit. An approximation algorithm should gracefully adapt to situation where a perfect solution requiring the only theoretical minimum of wildcards is not possible.

To achieve this goal we designed a greedy algorithm. The idea is to set one bit after another for every in a way that brings the solution closer to requiring only one wildcard per link. The greedy algorithm will iterate through all bits and set one bit after another for all paths. Initially all paths have no bits set and are indistinguishable. 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.

Uniformly select an edge $e$. Determine the bits that all paths that use the edge have in common. Using these bits as wildcard to all other edges. If any of the paths of an edge matches the wildcard the edge is put in the set $U$. The set contains the edges with paths that cannot distinguished from $e$ with the wildcard. If the set of indistinguishable edges $U$ is empty, move to the next edge. Otherwise set the unset bits of the paths on $e$ and the paths of the edges in $U$ so that the size $U$ is minimized. The algorithm will be run until either the set of indistinguishable edges is empty for every edge or when the number of allowed bits is reached.

Consider Figure 4 where the paths have partial path labels. A wildcard using the common bits of edge $e_1$ also matches one path of $e_2$ and $e_3$. Adding $0$ to the paths of $e_1$ and $1$ to the paths of $e_2$ and $e_3$ allows the wildcard 1XX01 to only match 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 non optimal 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 and 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 not common between all paths.

Then we choose the bit which minimizes $U_1$ and $U_0$. We repeat this step until for all wildcards $w_i$ of the path the sets $U_i$ are empty.

## V. Evaluation

Our evalution consists of two parts. The first part consists of an OpenFlow network with multiple hosts and different operating system (Windows, Linux, Mac OS X, Cisco IOS) to confirm that our method of using ARP caches to label outgoing packets (Section 3) works as anticipated. Even though methods are confirm to the standard the behavior of the network is unusual from the operating system perspective. As test bed we built an OpenFlow network using Mininet [6] and connected various virtual and physical hosts to it.

We installed flow rules that rewrote the modified destination mac addresses back to the original mac addresses and intercepted the ARP request with our controller and send modified ARP replies back.

Our finding 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 arbitray. Or from an Ethernet layer centric view, the operating systems do not make assumption about the network that violate the possibility of asymmetric routing on a Ethernet layer.

The second part was a simulation to evaluate the possibility of reducing the number of needed flow table entries by using the greedy algorithm described in subsection 4. As input we built CLOS and fat tree data center topologies. For the paths we calculated between all switches the shortest paths and kept all paths with minimal costs giving the possibility to choose between multiple paths between two end host by answering with the MAC address of the chosen path for the ARP reply.

## VI. Related work

Reducing the FIB size has been studied for various different scenarios and requirements. A very general approach to solve the problem are compact routing algorithms, which are designed for arbitrary networks. For example, the compact routing scheme of [16] has a table size of $\tilde{\mathcal{O}}(n^{1/2})$ with a stretch factor of 3 is achieved.

protocol proposal also 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 policy routing possible using BGP. Pathlet routing achieves this flexibility by adding labels to the packets that encode parts of the path and allows each AS to decided itself to decide trade-off between number of FIB entries and complexity of the implemented routing policy. Even though this is aimed at a different problem, the question arises if these FIB reduction techniques (or generally FIB reduction techniques aimed at inter-AS communication [17], [3]) can be applied to the data-center FIB reduction. The main similarity is that in both cases the actual forwarded elements are unstructured, mac addresses in the data center and IP prefixes in the inter-AS communication. In both scenarios labels of some kind are used to aggregate and structure the unstructured forwarding elements. A key difference is that data center routing does not involve multiple parties with different goals. These approaches focus on possible scenarios and techniques designed for future routers and Internet protocols. Therefore these labels are not designed to work on hardware or require hardware capabilities not present in current data center hardware.

Data center networks differ a lot from the Internet topology and are usually highly structured. For a modern data center network the FIB table consists mainly of basic Layer 2 forwarding (Ethernet) entries

as lookup key and the output port as forwarding action. Since Ethernet addresses of hosts are not structured a FIB table for a data center switch needs one entry per host in the network. Especially in large networks this number of entries is not feasible. To make it feasible for a switch to hold the full forwarding table the number of entries in the table has to be reduced. Without coordination compressing the entries has to be done individually on each switch. This is generally difficult, since the link layer addresses are unstructured and achieving a good compression ratio also requires modifying the hardware. [15].

Instead of matching with bitwise wildcards the approach in [14] uses Bloom filters for matching addresses. The approach is targeted at networks which have a relatively large number of alternative paths for each communication pair like CLOS networks. Matching using bloom filters can have false positives when matching packets. By not using the paths which trigger false positives the approaches avoids this problem. For this to work the networks needs to have multiple pathes for every communication pair. This makes the approach only applicable for network with heavy multi pathing like a CLOS network. The matching by bloom filters is furthermore not possible without modification of the existing (OpenFlow) software and hardware. These two requirements are not required when using our approach.

With coordination by either implementing a distributed algorithm or using a central network control instance like in SDNs, reduction of the FIB table size can be achieved by doing a structured assignment of the link layer (MAC) addresses to hosts and switches [5], [11]. The MAC addresses usually either encodes a position of the switch in the network or the MAC address is used as a routing element. Encoding the path into the MAC address is an example for that. These approaches either assume a certain network structure to work and sometimes require matching/rewriting capabilities.

There are few similarities between our protocol and PortLand. Portland assign each switch a hierarchical level of either core, aggregation or edge. The format of the pseudo mac used by PortLand is fixed and encodes the hierarchical structure. The pseudo MAC is used as a label encoding the destination host. Portland will rewrite the source and the destination MAC address on the ingress and egress TOR switches. In contrast we have shown that rewriting the source address is unnecessary and that the destination MAC address can be used as true label and not only as 1:1 label for the actual destination MAC address.

The Portland and VL2 approaches can also benefit from our technique of using MAC address as labels. In the case of the PortLand protocol the pseudo MAC can be directly stored in the ARP tables of the connected hosts and does not need to be rewritten by the ingress switch.

First hop redundancy protocol (FHRP) (like HSRP[7] or VRRP[9]) multiple routers 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 clients always has a valid entry as long as one routers is still working. When forwarding a packet to a clients a router will use its own MAC address.

## VII. Conclusion

We have shown that our techniques for reducing the number of needed flow table in an software defined network are viable and working options. The new paradigm of software defined networks allows to reinterpret network concepts and header fields and use them for other purposes. 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. We have further shown that using MAC addresses as forwarding labels very small FIB tables can be achieved. The technique also enables us to improve other protocols.

## ACKNOWLEDGMENT

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

## 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] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. *SIGCOMM Comput. Commun. Rev.*, 39(4):51–62, Aug. 2009.

[6] 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.

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

[8] C. Meiners, A. Liu, and E. Torng. Bit weaving: A non-prefix approach to compressing packet classifiers in tcams. In *Network Protocols, 2009. ICNP 2009. 17th IEEE International Conference on*, pages 93–102, 2009.

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

[10] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor Discovery for IP version 6 (IPv6). RFC 4861 (Draft Standard), Sept. 2007. Updated by RFCs 5942, 6980, 7048.

[11] 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.

[12] D. Plummer. Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 826 (INTERNET STANDARD), Nov. 1982. Updated by RFCs 5227, 5494.

[13] G. Rétvári, J. Tapolcai, A. Kőrösi, A. Majdán, and Z. Heszberger. Compressing ip forwarding tables: Towards entropy bounds and beyond. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 111–122, New York, NY, USA, 2013. ACM.

[14] 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.

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

[16] 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.

[17] 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.