Efficient FIB Representations on Distributed Platforms

Kirill Kogan, Sergey I. Nikolenko, Patrick Eugster, Alexander Shalimov, Ori Rottenstreich

***Abstract*—The Internet routing ecosystem is facing substantial scalability challenges due to continuous, significant growth of the state represented in the data plane. Distributed switch architectures introduce additional constraints on efficiency of implementations from both lookup time and memory footprint perspectives. In this work we explore efficient FIB representations in common distributed switch architectures. Our approach introduces substantial savings in memory footprint transparently for existing hardware. Our results are supported by an extensive simulation study on real IPv4 and IPv6 FIBs.**

I. Introduction

The Internet routing infrastructure is widely acknowledged to be facing severe challenges in the long haul [1], [2]. While these challenges may not be critical for the control plane, the data plane is bogged down by the rapid expansion of forwarding tables constituting forwarding information bases (FIBs), a largely unsolved problem to date [3], [4].

Efficient FIB representations in the data plane become even more important with the advent of IPv6 since most methods that efficiently represent IP-based FIBs do not scale well to IPv6 due to its significantly larger 128-bit address width [5]–[8]. Distributed systems introduce additional complexities in FIB representations.

In this work, we consider the problem of representing FIBs on distributed platforms [9], [10], where several line-cards (LCs) are interconnected by a switching fabric (see Fig. 1b). Each ingress (RX) LC must separately maintain a FIB table (or parts of one) to forward traffic to the correct egress (TX) LC which transmits it further over an output port. The fundamental question here is how to efficiently implement a FIB table (Fig. 1a) across such a distributed switching platform.

Currently, there are two major types of FIB implementations for such distributed platforms: *one-stage* and *two-stage* forwarding. In the former, pointers to output ports are already found on ingress LCs (see Fig. 2a). Later, on an egress LC, the corresponding traffic is encapsulated based on the pointer coming from the ingress LC [9], so every ingress LC should contain information about “routable” prefixes of all egress LCs, whilst egress LCs encapsulate packets based on the pointer to the output port from the ingress LC. A major drawback of this representation is that maintaining

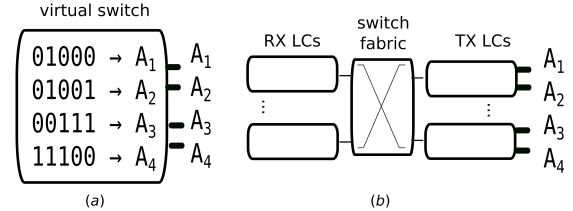


Fig. 1. How to represent FIB table (a) on distributed switching platform (b).

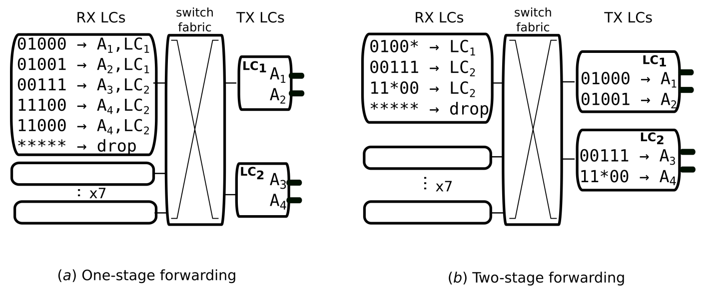


Fig. 2. One- and two-stage forwarding implemented on existing distributed platforms [10], [11]; lookups are done on all address bits.

prefixes mapped to output ports of all egress LCs on every ingress LC is not scalable. While such one-stage forwarding representations do not scale well, they are simple and require only dummy processing on egress LCs (processing pipelines are usually doubled to isolate ingress and egress traffic). For example, different generations of Cisco C12000 [9] LCs implement one-stage forwarding.

In contrast to one-stage forwarding, where output ports are already computed at ingress LCs, ingress LCs in two-stage forwarding only have to choose which egress LC to send a packet to (see Fig. 2b). Now egress LCs have to choose a specific output port by running an additional FIB lookup on the whole destination address for every packet. This allows to mitigate memory constraints on ingress LCs, and each egress LC now contains only prefixes corresponding to its output ports. For a specific router example, different generations of Cisco CRS-1 [9] LCs implement two-stage forwarding.

FIB tables can be represented in different ways. Informally, we say that two representations $\mathcal{K}$ and $\mathcal{K}'$ are *equivalent* if for any input header both $\mathcal{K}$ and $\mathcal{K}'$ yield the same action. Different equivalent FIB representations can optimize distinct objectives, e.g., minimize required memory or number of entries [7], [12]. The above-mentioned one- and two-stage forwarding representations are equivalent to FIB representation on a virtual switch (see Fig. 1a) and are currently used in real-life distributed platforms [9], [10]. Note that all bits

K. Kogan is with IMDEA Networks Institute; e-mail: kirill.kogan@imdea.org. S.I. Nikolenko is with National Research University Higher School of Economics and Steklov Mathematical Institute at St. Petersburg; e-mail: sergey@logic.pdmi.ras.ru. P. Eugster is with TU Darmstadt and Purdue University; e-mail: peugster@dsp.tu-darmstadt.de. A. Shalimov is with Lomonosov Moscow State University; e-mail: alex.shalimov@gmail.com. O. Rottenstreich is with Princeton University; e-mail: orir@cs.princeton.edu.

of addresses are used in both one- and two-stage forwarding which extremely limits efficient implementations, especially on ingress LCs.

The primary contribution of this work is to exploit structural FIB properties to balance bits that are looked up between ingress and egress LCs, minimizing required memory and lookup time. This is not a specific classifier implementation; instead, we propose an *abstraction layer* that defines a subset of bit indices that should be involved in the lookup process on ingress and egress LCs respectively. As a result, a classifier based on these bit indices can be transparently represented by other schemes (e.g., [7], [13]). Our abstraction, based on the proposed structural properties, generalizes beyond IPv4, IPv6, or flow-based FIB representations. It does not require changes to network-wide control plane signaling and underlying network infrastructure: FIBs are given and computed by any existing routing protocol. In contrast to compact routing [2], our approach does not affect routing decisions based on requested address space, yet our methods can achieve significant memory savings without increasing lookup time (actually improving it) and are applicable even if the input FIBs have already been optimized. As a byproduct, we show a counter-intuitive result: IPv6 FIBs can be implemented on existing IPv4 or MPLS hardware- or software-based FIB implementations without increasing lookup time or memory requirements or requiring changes in hardware. We do not limit ourselves to prefix-based classifiers with longest-prefix-match (LPM) priorities either and consider the general case of classifiers applicable to clean slate architectures. We also demonstrate how to achieve additional savings when network-wide representations are considered.

This paper significantly extends a preliminary report [14] with revised Section IV, reworked and extended with new experimental results Section V, Section VI written from scratch, and extended introductory and survey parts. The paper is organized as follows. Section II introduces the basic model. Section III introduces the notion of MATCH EQUIVALENCE and shows how to use three flavors of MATCH EQUIVALENCE for efficient FIB implementations on distributed systems and extend them to network-wide representations. In Section IV we propose algorithms for memory minimization of FIBs based on MATCH EQUIVALENCE; Section V demonstrates 50-80% savings over *optimized* equivalent solutions achieved by state-of-the-art Boolean minimization techniques on real FIBs, both IPv4 and IPv6. Section VI discusses dynamic updates for the considered representations. In Section VII we compare lookup times of the proposed representations versus baseline IP and IPv6 implementions in the Intel Data Plane Development Kit (DPDK) [15]. Though our representations lead to substantial memory savings, they are also approximately twice faster than the baseline implementation of IPv6 in DPDK and on par with the baseline IP implementation. Section VIII discusses related work, and we conclude with Section IX.

II. Model Description

A packet *header* $H = (h_1, \cdots, h_w)$ is a sequence of w bits from a packet and internal databases of network elements, where each bit h_i of H has a value of zero or one, $1 \le i \le w$. We denote by W the ordered set of w *indices* of the bits in headers (i.e., $(1, ..., w)$). A *classifier* $\mathcal{K} = (R_1, ..., R_N)$ is an ordered set of rules, where each rule R_i consists of a *filter* F_i and a pointer to the corresponding *action* A_i. A filter F is an ordered set of w values corresponding to the bits in headers. Possible values are 0, 1, and $*$ ("don't care"). A header H *matches* a filter F if for any bit of H, the corresponding bit of F has the same value or $*$. A header H matches a rule R_i if R_i's filter is matched. The set of rules has non-cyclic ordering $\prec$; if a header matches both R_i and R_j for $R_i \prec R_j$, the action of rule R_i is applied. We say that two filters F and F' of a classifier *intersect* if there is at least one header that matches both F and F'; otherwise, F and F' are *disjoint*. For instance, for $w = 4$, $F_1 = (1\ 0\ 0\ *)$, $F_2 = (0\ 1\ *\ *)$, and $F_3 = (1\ *\ *\ *)$, F_1 and F_3 intersect (for instance, the header $(1\ 0\ 0\ 0)$ matches both filters), while F_1 and F_2 are disjoint. A classifier is called *complete* iff it matches any possible header. Any incomplete classifier $\mathcal{K}$ can be transformed to a complete one by appending $\mathcal{K}$ with a *default* rule that matches all unmatched headers.

Let B be a set of bit indices, $B \subseteq W$, referring to a subset of the bits in packet headers. For a header H, we denote by H^B the (sub)header of $|B|$ bits obtained by taking only bits of H (in their original order in W) with indices in B. Likewise, for a rule R or a filter F we denote by R^B and F^B, respectively, the (sub)rule and the (sub)filter defined on $|B|$ bits by removing the bits from $W \setminus B$. Finally, for a classifier $\mathcal{K} = (R_1, ..., R_N)$, let $\mathcal{K}^B$ be the (sub)classifier obtained from $\mathcal{K}$ by replacing each rule R in the classifier by R^B. The notions of matching, intersection, and disjointness carry over to such subsequences. Similarly, for a header H, we denote by H^{-B} the header of $w - |B|$ bits obtained from H by considering only bits with indices in $W \setminus B$. Likewise, let R^{-B} and F^{-B} be the rule and the filter, respectively, defined on $w - |B|$ bits by eliminating the requirement on the bits in B for a rule R or its filter F. Let $\mathcal{K}^{-B}$ be the classifier obtained by replacing each rule R in a classifier $\mathcal{K}$ by R^{-B}. We denote the set of all filters in $\mathcal{K}$ based on the ordered set of bit indices $B \subseteq W$ by $\mathcal{F}^B$; $\mathcal{F} = \mathcal{F}^W$. We denote the set of all actions in $\mathcal{K}$ by $\mathcal{A}$.

Consider a classifier $\mathcal{K}$ with a classification function $f : \{0,1\}^w \to \mathcal{A}$. For a set of bit indices $B \subseteq W$, we say that a classifier $\mathcal{K}^{-B}$ with classification function $g : \{0,1\}^{w-|B|} \to \mathcal{A}$ is an *equivalent* representation of $\mathcal{K}$ if $f(H) = g(H^{-B})$ for every header $H \in \{0,1\}^w$.

III. FIB Representations

Before we introduce representations based on the proposed structural properties formally, we present an illustrative example (Fig. 1a) to motivate our contributions.

A. Traditional Forwarding Representations

Consider a virtual switch (Fig. 1b) representing a distributed system. The major drawback of one- and two-stage forwarding approaches is the fact that all bits of a lookup address are represented in FIB tables. In particular, in the case of one-stage forwarding (Fig. 2a), every ingress LC requires five 5-bit

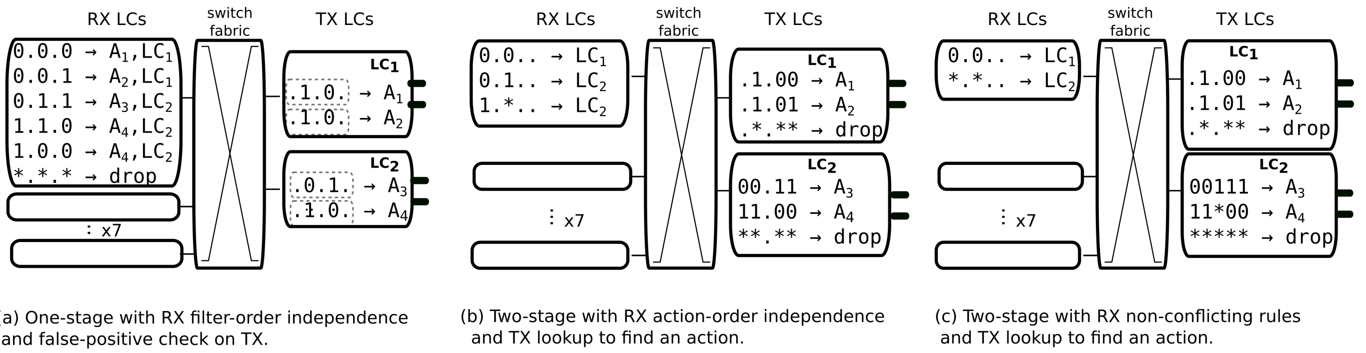


Fig. 3. Equivalent FIB representations with reduced address bits.

entries (not including a default entry) without additional FIB table on egress cards (Fig. 2a), so the total memory needed for eight slots is $6 \times 5 \times 8 = 240$ bits, and there is a single FIB lookup per packet that is done on ingress LCs.

In the two-stage forwarding case (Fig. 2b, in our recurring example), actions A_1 and A_2 for output ports are unified to a single "action" LC_1, and actions for output ports A_3 and A_4 to LC_2 on ingress LCs. This allows to further compress FIB entries on ingress LCs, uniting for instance prefixes 01000 and 01001 with the same action LC_1 into a filter $0100*$. Now every ingress LC requires three 5-bit entries (not including a default entry). In addition each egress LC_1 and LC_2 requires two 5-bit entries. The total memory needed on all ingress and egress LCs in our example is $4 \times 5 \times 8 + 2 \times 5 \times 2 = 180$ bits.

B. Balancing Bits between Ingress and Egress

Note that interconnecting switch fabrics should support full linerate for any combination of ingress and egress LCs, and therefore should not become a bottleneck. Also FIB lookups on egress LCs in the case of two-stage forwarding (or dereferencing of a pointer found on ingress in the case of one-stage forwarding) is usually the first thing to be completed on egress LCs to determine further processing of packets (e.g., the pipeline architecture of packet processing engine in Cisco C12000 router [16]). As long as these invariants are maintained we can move away from equivalent representations on the LC level to equivalent representations in the boundaries of the same distributed system by balancing between evaluating bits of a represented FIB table on ingress LCs versus egress LCs. In the following, we propose several structural properties of classifiers and show how to exploit them to find more efficient representations of FIBs on distributed switching platforms.

C. Match Equivalence

When we reduce a classifier's width, we often get a classifier that does not preserve the semantics (we do not have *strict* equivalence). However, removing a given bit index from a classifier is in essence equivalent to changing the bit's value to $*$ in all filters, so reducing width cannot decrease the set of headers covered by filters. Hence, for this relaxation we just need to guarantee a correct match of headers that are matched by the original classifier. We introduce a novel property, MATCH EQUIVALENCE, to improve the efficiency of incomplete classifiers. This property relaxes the classification function by keeping only bit indices that preserve the original classifier $\mathcal{K}$'s action for any header *matched* by $\mathcal{K}$. Formally, an incomplete classifier $\mathcal{K}^B$ is MATCH EQUIVALENT (ME) to a classifier $\mathcal{K}^W$, $B \subseteq W$, if the action for any matched header H^W in $\mathcal{K}^W$ coincides with the action for the corresponding header H^B in $\mathcal{K}^B$. If H^W has no match in $\mathcal{K}^W$ then H^B can either be matched or remain unmatched in $\mathcal{K}^B$[1].

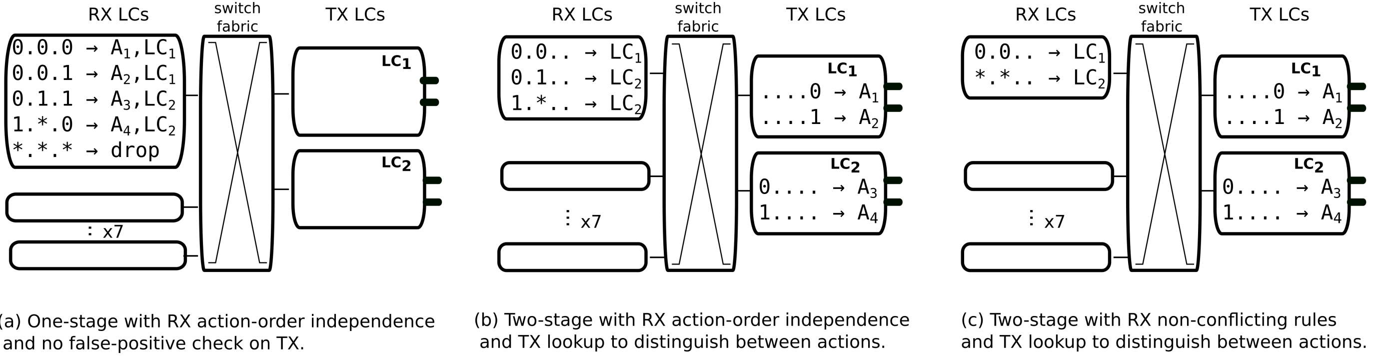
Problem 1 (Minimal MATCH EQUIVALENCE)**.** *For a given classifier $\mathcal{K}$, find a* MATCH EQUIVALENT *classifier to $\mathcal{K}$ whose filters require minimal memory (in bits).*

This definition of MATCH EQUIVALENCE does not directly yield algorithms for the MINME problem. Next we propose different flavors of MATCH EQUIVALENCE based on three structural properties. In Section IV we will show what can be done if a classifier does not implement these properties. In Section V we will compare the effects of these structural properties on the desired objective.

D. Filter-order Independence

First we consider a family of MATCH EQUIVALENT classifiers that exploit *filter-order independence*. Classifiers whose filters are all pairwise disjoint (that do not match the same headers) are *filter-order-independent*. Intuitively, representations based on filter-order independence (alone) do not take into account the number of different actions. Kogan et al. [17] originally demonstrated the effect of this characteristic on classifiers whose fields are represented by ranges; the impact of filter-order independence on FIB representations was left unaddressed. Filter-order independence guarantees that only a single filter is matched, so to keep this property, MATCH EQUIVALENT classifiers on ingress linecards should preserve filter-order independence. In addition, to build an *equivalent* solution we need to check on egress LCs if the remaining bits (that did not participate in the classification process on ingress) of the packet still match an admissible packet (see Fig. 3a). If yes, the forwarded packet is sent out through the corresponding output port, otherwise it is dropped, similarly as

[1] MATCH EQUIVALENCE can also be applied to complete classifiers when the action of the default entry drops all unmatched traffic and does not implement routing decisions. In this case we can compute a MATCH EQUIVALENT classifier on $\mathcal{K}$ without the default entry and later append the default entry back to the result.



(a) One-stage with RX action-order independence and no false-positive check on TX.

(b) Two-stage with RX action-order independence and TX lookup to distinguish between actions.

(c) Two-stage with RX non-conflicting rules and TX lookup to distinguish between actions.

Fig. 4. Non-equivalent FIB representations with reduced address bits.

it is done by representations in Fig 2. Observe that in this case there is no lookup table on egress LCs and the remaining bits are fetched from the pointer found on ingress, so filter-order independence has a good fit for one-stage forwarding. Every packet matching the ingress FIB (except the default entry) in Fig 2a (that implements one-stage forwarding) applies the same action as in the compressed ingress FIB in Fig 3a. But now packets previously matched by the default entry in the ingress FIB in Fig 2a may be matched by the compressed ingress FIB in Fig 3a and are dropped only on egress LCs. Still this representation is equivalent in the system boundaries and allows to balance lookup time and memory requirements between ingress and egress LCs. In the running example the prefixes of a given FIB are filter-order independent and the FIB on every ingress LC can be reduced to only five 3-bit entries (see Fig. 3a) (not including a default entry), with a total memory requirement for this representation being $6 \times 3 \times 8 + 2 \times 2 \times 2 = 152$ bits. As a significant side effect, this representation allows to represent IPv6 on already existing IP or even MPLS ingress LCs. We demonstrate the feasibility of this representation in Section V for real IPv6 FIBs. Note that filter-order independence can be too restrictive; it is unnecessary to require this property from rules with the same action.

E. Action-order Independence

The major drawback of filter-order independence is that it can reduce only a number of bit-identities that are involved in FIB representations on ingress LCs but cannot reduce the number of entries in these tables on ingress.

To address this limitation we introduce a family of MATCH EQUIVALENT classifiers that exploit *action-order independence*. This characteristic is based on the observation that a classification result really is the *action* of the rule with the highest priority, rather than the filter associated with this rule. Two rules with filters F_1^B and F_2^B are *action-order-independent* if either (a) they have the same action or (b) F_1^B and F_2^B are disjoint; otherwise, we say that two rules are *action-order-dependent*. Similarly, a classifier $\mathcal{K}$ as a whole is called action-order-independent if every pair of its rules are action-order-independent; otherwise $\mathcal{K}$ is action-order-dependent. Conversely, a classifier $\mathcal{K}'$ that implements action-order independence on a subset $B \subset W$ of bit indices of an originally action-order-independent classifier $\mathcal{K}$ matches the same headers as $\mathcal{K}$, and as a result $\mathcal{K}'$ is MATCH EQUIVALENT to $\mathcal{K}$ with respect to B.

The action-order independence allows to further reduce the number of entries for filters with the same action. In this case we can reduce both the number of entries and the number of bit-identities participating in the FIB lookup on ingress. The price for this is a slightly more complex "false-positive" check and thus processing on the egress LCs, that in contrast to filter-order independence requires a FIB lookup table based on a subset of bit-identities (see Figure 3b). Note that the two-stage forwarding approach with action-order independence creates an additional reduction in the number of entries on ingress since a number of actions is reduced to a number of different LCs. In our example the memory requirements for this representation type is $3 \times 2 \times 8 + 3 \times 3 + 3 \times 4 = 69$ bits[2]. Since this representation is more general than the previous one it supports an even better balance of memory and lookup times between ingress and egress LCs. As we will show in a comprehensive comparison on real classifiers shortly in Section V, it suffices to achieve action-order independence on a subset of bit indices to find much more space-efficient representations of classifiers than representations that exploit filter-order independence.

F. Non-conflicting Rules

While exploiting action-order independence to implement MATCH EQUIVALENCE has a huge potential for memory savings (as demonstrated by our evaluations in Section V), it becomes apparent that we need to better understand the applicability and expression of MATCH EQUIVALENCE. To that end we suggest to express the third type of MATCH EQUIVALENCE through the notion of *conflicting* rules. Two rules R_1^X and R_2^X with different actions, $R_1^X \prec R_2^X$, are *conflicting* with respect to bit indices $B \subset X \subseteq W$ if there is a header H^X matching R_2^X that is not matched by R_1^X and R_1^B matches H^B.

Example 1. In the following classifier with $|X| = |W| = 3$

$$\begin{array}{lcccl} & \#1 & \#2 & \#3 & \\ R_1^X & (* & 1 & 0) & \to A_1 \\ R_2^X & (1 & 1 & *) & \to A_2 \end{array}$$

[2]Note that only in this specific example no default entry is required on ingress LCs since all combinations of values are already covered.

we have that R_1^X and R_2^X are conflicting with respect to $B = \{1, 2\}$ since R_1^B matches $H^B = (1\ 1)$ but R_1^X does not match $H^X = (1\ 1\ 1)$ when R_1^B matches H^B and yet it matches R_2^X.

On the other hand, for $B = \{2, 3\}$, the rules R_1^X and R_2^X are not conflicting with respect to B.

$$\begin{array}{llll} & \#2 & \#3 & \\ R_1^B & (1 & 0) & \to A_1 \\ R_2^B & (1 & *) & \to A_2 \end{array}$$

A classifier $\mathcal{K}^B$ is MATCH EQUIVALENT to a classifier $\mathcal{K}^X$ with respect to B, $B \subset X$, if no two rules $R_1^X \prec R_2^X$ in $\mathcal{K}^X$ with different actions are conflicting with respect to B.

For our running example the FIB representation based on non-conflicting rules on each ingress LC requires two 2-bit entries (see Figure 3c) with the total memory requirement of $2 \times 2 \times 8 + 3 \times 3 + 3 \times 5 = 56$ bits.

The proposed representations based on three types of MATCH EQUIVALENCE are applicable to FIBs based on multi-field rules with general priorities such as in OpenFlow. Complexity bounds derived from computational geometry imply that a software-based packet classifier with N rules and $k \geq 3$ fields uses either $O(N^k)$ space and $O(\log N)$ time or $O(N)$ space and $O(\log^{k-1} N)$ time [18], [19]. So in the general case adding new fields that do not have exact values in all rules can lead to significant overhead in either memory requirements or lookup time.

Consequences of all three types of MATCH EQUIVALENCE that we consider are counter-intuitive. For equivalent representations on the LC level, more bit identities in given classifiers with a fixed number of filters implies bigger memory footprint (e.g., IPv6 or OpenFlow versus IP). This is not the case for MATCH EQUIVALENCE. Furthermore, MATCH EQUIVALENT FIB tables with A actions based on action-order independence and non-conflicting rules has $A \log A$ minimal memory requirements, since we need at least one filter for every action, and $\log A$ bits are required to distinguish among A actions. With MATCH EQUIVALENCE, IPv6 and OpenFlow have better chances than IP to find bit indexes to distinguish between actions. This allows us not only to significantly minimize memory footprint versus optimal equivalent representations on the LC level, but also to implement ingress IPv6-based FIBs or other clean-slate solutions (e.g., OpenFlow) on existing IPv4 or even MPLS infrastructures without increasing memory requirements and lookup time, transparently to the hardware. We will demonstrate this effect in Section V. Fig. 3 summarizes all proposed representations which are at the system level equivalent to the one- and two-stage forwarding representations in Fig. 2 and introduce substantial savings on ingress and egress LCs.

G. Network-wide Equivalent Representations

Compact routing is an approach that can reduce FIB sizes for already optimized forwarding tables (for a given objective) by *stretching* the "shortest-path" requirements [2]. Naturally, there is a trade-off between the stretch factor and FIB sizes. Notably, certain compact routing schemes tailored to trees or grids scale logarithmically with the network size [2]. According to [2], there is no widely-used routing protocol with stretch factor larger than one: link-state (OSPF or ISIS), distance-vector (RIP or IGRP), and path-vector (BGP) protocols are all based on shortest-path routing (stretch factor 1).

So far all proposed representations operate on a given (possibly already optimized) FIB and implement EQUIVALENCE within the confines of a single network element. As a result, as opposed to compact routing, these schemes do not require changes in control plane signaling and do not affect the "quality" of implemented routing decisions. In this part we generalize previously proposed FIB representations and discuss EQUIVALENCE with network-wide resolution; this can yield further savings in memory footprint but will require changes in control plane signaling and possibly of the network-wide view. For instance, this approach can fit well for software-defined networking with centralized management.

We return to our original question of how to represent the FIB table of a virtual switch from Fig. 1a on a distributed system in Fig. 1b, but now the distributed system is also virtual and can represent the whole network. Various prior works represent the whole network as a single virtual switch (e.g., [20], [21]). Clearly, equivalent representations at the LCs or system levels are applicable during mapping of this virtual switch to physical network elements.

But we can go further and decide to skip false-positive tests on egress LCs of the edge router A, ideally on egress LCs of relevant edge routers B and C facing core interfaces (see Fig 5)[3]. Fig. 4 summarizes non-equivalent solutions that allow to achieve further memory savings. Amazingly in the case of IPv6, we can exploit these non-equivalent representations to decrease the size of FIB tables on ingress and egress LCs of router A, and for instance, representations based on one-stage forwarding with filter-order independence on ingress of the routers B and C[4]. A combination of these two representations allows to avoid complex 128-bit representations on routers A, B, and C. For instance, the engine 3 LC family of Cisco C12000 [9] implement IP FIB as a three level 16-8-8 bit-trie, where every level requires exponential memory. Usually these schemes can clearly not be extended for IPv6. As a result engine 3 LCs used TCAM for IPv6 that was sharable with other services such as QoS, ACL, etc. So IPv6 scalability was very limited for this LC family. The following engine 5 LC family already had four TCAMs with 10^6 of 72-bit entries on every ingress and egress LC to support the desired level of IPv6 scalability, significantly increasing cost and power requirements. The proposed network-wide FIB representations can significantly reduce these resource requirements of representing edge routers, and allow deploying IPv6 without involving TCAMs on existing IP and MPLS infrastructure. Observe that these network-wide representations can be considered as equivalent in the boundaries of the whole network since in the worst case some packets that were dropped in the router A with equivalent solutions from Fig. 2 and Fig. 3 now can be dropped on egress LCs of the routers B and C. In contrast to equivalent representations on the system level, where a

[3]The core network in Fig 5 can run IP/IPv6 or MPLS.

[4]Note that in this case egress LCs of these routers do not need to run FIB lookups just a false-positive check based on the "remaining bits" of IPv6 address based on the found on ingress pointer.

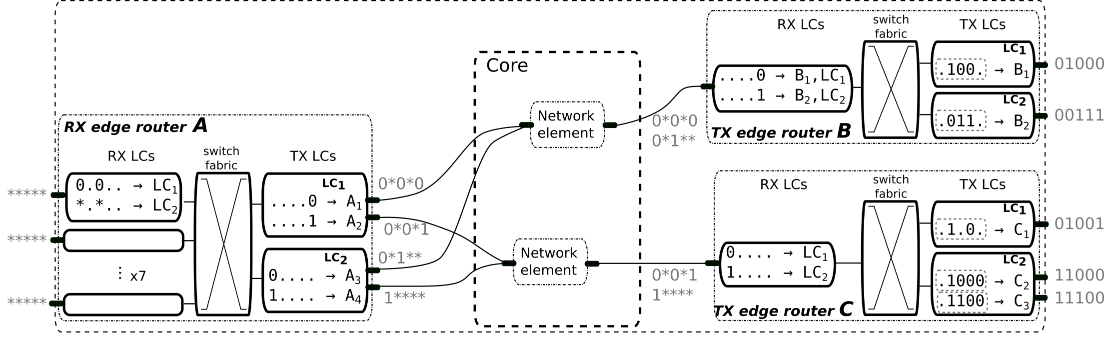


Fig. 5. A sample network-wide FIB representation.

switching fabric is not a bottleneck, we understand that there is a fundamental tradeoff between memory requirements on edge routers and potentially "wasted" core bandwidth for traffic that is now dropped only at the egress of edge routers B and C but want to bring these alternatives for consideration. All proposed methods that exploit proposed structural properties are valid also for general classifiers as in OpenFlow.

Having illustrated the power of the proposed representations, let us now proceed with algorithms that compute them.

IV. Algorithms for Match Equivalence

In this section we propose algorithms that compute optimized MATCH EQUIVALENT classifiers that are applicable to all three considered structural properties: filter-order independence, action-order independence, and non-conflicting rules.

Removing redundant entries and bit indices further reduces the memory footprint bottleneck on ingress and egress linecards. Instead of proprietary algorithms for this purpose, we suggest to use heuristics based on Boolean minimization techniques for ordered AC_3^0 circuits [22] in all considered FIB representations (including traditional one- and two-stage forwarding). In what follows we denote this building block by SE (*strict* equivalence); SE computes an equivalent representation in minimal memory. Unfortunately, some of the more sophisticated SE algorithms have complexity $O(N^3 \cdot w)$, where N is the number of rules and w is their width[5]. For this reason, in our evaluation in Section V, we use a simplified version of SE that exploits only a part of the SE heuristics and runs in $O(N^2 \cdot w)$ time. In preliminary evaluation on hundreds of FIBs, we did not find any significant difference in memory reduction between the simplified and full versions of SE for all considered representations (including traditional one- and two-stage forwarding). Here, we use SE in two major cases: (1) to remove redundant filters and bit indices; (2) to check potential memory savings while removing a specific bit index.

A. MATCH EQUIVALENCE *on all Rules*

We start with Algorithm 2 (which uses Algorithm 1) that tests whether $\mathcal{F}^B$ is MATCH EQUIVALENT (ISME) to a given $\mathcal{F}^X$, where B a subset of bit indices X. We say that two rules *break a structural property* if they are filter-order-dependent, action-order-dependent, or conflicting. In this way all proposed algorithms can be used for all structural properties[6]. In Algorithm 2, we have denoted by $\textsc{Succ}(\mathcal{F}, F) = \{F' \in \mathcal{F} \mid F \prec F'\}$, i.e., the set of all rules in $\mathcal{F}$ after F.

[5] A possible SE implementation (OPTIMIZECLASSIFIER) with examples can be found in [23].

Algorithm 1 FINDBREAKERS(F,$\mathcal{F}$,B)

```
1: A ← ∅
2: for every F′ ∈ 𝓕 starting from F do
     ▷ two filters that break a struct. property can be
     ▷ filter- or action-order-dependent, or conflicting rules
3:   if F and F′ break a structural property with respect to B then
4:     A ← A ∪ {F′}
5: return A
```

Algorithm 2 ISME($\mathcal{F}^X$,B)

```
1: for every F_1^X ∈ 𝓕^X do
2:   if FINDBREAKERS(F_1^X, SUCC(𝓕^X, F_1^X), B) ≠ ∅ then
3:     return False
4: return True
```

So all algorithms proposed below will check if removing a certain bit still retains MATCH EQUIVALENCE, and will differ in how they choose which bit to remove.

Algorithm 3 considers bit indices that can be removed while preserving MATCH EQUIVALENCE. At each step, it removes one bit index from all filters; the bit index is selected to maximize the memory savings obtained by the reduction (Lines 6-7). The algorithm continues as long as there is a bit index that can be removed while preserving MATCH EQUIVALENCE. In Line 9, SE removes rules that collapse after reducing a subset of bit indices.

Example 2. Consider the following classifier:

$$\begin{array}{ccccl} \#1 & \#2 & \#3 & \#4 & \\ (0 & 1 & 0 & 0) & \to A_1 \\ (0 & 1 & 1 & 1) & \to A_2 \\ (1 & 1 & 1 & 0) & \to A_2 \end{array}$$

The proposed heuristic MINME() will consider three bits that can be potentially removed in the first step: #1, #2, and #4. Among these bits, removing #2 does not lead to further memory reductions, while removing, say, #1 leads to $F_2^{-\{1\}} = (1\ 1\ 1)$ and $F_3^{-\{1\}} = (1\ 1\ 0)$, which can now be

[6] Actually, Algorithm 2 tests if all different pairs of rules do not break a structural property on a specific subset of bit indices. Algorithm 2 is implemented through Algorithm 1 to increase reuse with other algorithms in this section.

Algorithm 3 MINME $(\mathcal{K}^X)$

```
1: B ← X, I^B ← K^X
2: repeat
3:     B' ← B
4:     for each j ∈ B do
5:         if IsME(I^{B\{j}}, B \ {j}) then
   ▷ remove j-th bit index whose removal lead to the minimal size reduction
6:             j* ← arg min_j SE(I^{B\{j}})
7:             B ← B \ {j*}
8: until B = B'
   ▷ reduce collapsing rules after removed bit indices
9: I^B ← SE(I^B)
10: return B, I^B
```

united by resolution to $F' = (1\ 1\ *)$; similarly, removing #4 leads to $F' = (*\ 1\ 1)$ on bits 1-3. Therefore, the heuristic chooses either #1 or #4, say #1, leading to the following MATCH EQUIVALENT classifier:

#2	#3	#4	
(1	0	0)	$\rightarrow A_1$
(1	1	$*$)	$\rightarrow A_2$

It is impossible to further reduce the set of rules (all rules have different actions now), so the proposed heuristic will not have any preference in subsequently removing bits #2 and #4; after removing them, we get the final representation that is based only on bit #3.

Property 1. *For a classifier with N rules and headers of w bits, the time complexity of* MINME() *is $O(N^2 \cdot w^3) + O(w^2) \cdot T(\text{SE})$, where $T(\text{SE})$ is the time complexity of the* SE *procedure.*

B. MATCH EQUIVALENCE *on a Subset of Rules*

We can further improve efficiency of representations and also deal with MATCH EQUIVALENT classifiers whose classification width is irreducible while preserving one of the structural properties. In particular, we propose to find a MATCH EQUIVALENT classifier on a *subset* of rules $\mathcal{I}$ of a given classifier $\mathcal{K}$ and moving "conflicting" rules of $\mathcal{I}$ to $\mathcal{D} = \mathcal{K} \setminus \mathcal{I}$, so that $\mathcal{I}^B$ is MATCH EQUIVALENT to $\mathcal{I}$ for some subset of bit indices $B \subseteq W$, and $\mathcal{D} = \mathcal{K} \setminus \mathcal{I}$ contains all other rules of the original classifier. The header of an incoming packet is matched in both groups of rules and the rule with the highest priority is chosen.

Example 3. In the following classifier

#1	#2	#3	#4	#5	#6	
(0	1	1	0	0	0)	$\rightarrow A_1$
(0	0	0	1	0	0)	$\rightarrow A_2$
(1	0	0	0	1	0)	$\rightarrow A_3$
(1	1	0	0	0	1)	$\rightarrow A_4$
($*$	$*$	0	0	0	0)	$\rightarrow A_5$

R_5 does not let us remove bits 3-6 because it conflicts with rules R_1-R_4 with respect to these bits. However, once we set $\mathcal{D} = \{R_5\}$, the remaining $\mathcal{I}^{\{1,2\}}$ becomes MATCH EQUIVALENT to the following classifier on fields $\{1, 2\}$: $(0\ 1) \rightarrow A_1$, $(0\ 0) \rightarrow A_2$, $(1\ 0) \rightarrow A_3$, $(1\ 1) \rightarrow A_4$.

Note that $\mathcal{D}$ is maintained in a traditional way, which can be expensive. Varying B we can find a feasible representation of $\mathcal{I}^B$ and reduce $\mathcal{D}$; applying an additional constraint on the maximal value of $|B|$ can reduce the size of $\mathcal{I}^B$. This additional constraint can allow to represent IPv6 on existing IP infrastructure. Splitting of an original classifier into $\mathcal{I}$ and $\mathcal{D}$ proves to be very fruitful for size reduction, as we will see shortly in Section V.

Problem 2 (MINME)**.** *Given a classifier $\mathcal{K}$ on a set of bits W and a positive number $l \le w$, find a subset of bits B, $B \subseteq W$, $|B| \le l$, and a subset of rules $\mathcal{I} \subseteq \mathcal{K}$ such that $\mathcal{I}^B \cup \mathcal{D}$ for $\mathcal{D} = \mathcal{K} \setminus \mathcal{I}$ is* MATCH EQUIVALENT *to $\mathcal{K}$ and minimizes the value of $|B| \cdot |\mathcal{I}| + w \cdot |\mathcal{D}|$.*

Algorithm 4 MINME $(\mathcal{K}^X, l)$

```
1: D_0 ← ∅, B ← X, K ← K^X, I ← K, w ← |B|
2: for i = 1..w do
3:     for each j ∈ B do
   ▷ find the set D_{i,j} of rules conflicting w.r.t. all bits except j
4:         D_{i,j} ← ∅
5:         for each R ∈ I do
6:             D_{i,j} ← D_{i,j} ∪ FINDBREAKERS(R, SUCC(K, R), B \ {j})
   ▷ choose the best j*, with the smallest number of conflicting rules
7:     j* ← arg min_j |D_{i,j}|
   ▷ add D_{ij} to D, update I_i, B, I, and K
8:     D_i ← SE(D_{i-1} ∪ D_{i,j*})
9:     I_i ← SE(K \ D_i)
10:    B ← B \ {j}, I ← I^B, K ← K^B
11: i* ← arg min_{i≥w-l} (w|D_i| + (w - i)|I_i|)
12: return i*, D_{i*}, I_{i*}
```

Algorithm 4 removes at least $w - l$ bits by greedily minimizing $|\mathcal{D}|$, which is equivalent to minimizing the value of $|B| \cdot |\mathcal{I}^B| + w \cdot |\mathcal{D}|$.

Property 2. *For a classifier with N rules and headers of w bits, the time complexity of* MINME() *is $O(N^2 \cdot w^3) + O(w^2) \cdot T(\text{SE})$, where $T(\text{SE})$ is the time complexity of the* SE *procedure.*

C. Exploiting Pseudo Parallelism

We can further generalize the idea of implementing MATCH EQUIVALENCE on a subset of rules to improve efficiency of FIB representations. Actually we can assign all rules into multiple disjoint groups, where every group obeys a structural property on a subset of bit indices. The number of such groups can be defined by the number β of parallel and serial lookups that can be performed at linerate, which is a property of a network element. Hence, we can further generalize Problem 2.

Problem 3 (PARMINME)**.** *Given a classifier $\mathcal{K}$ on bit indices W and two positive numbers $l \le w$ and β, assign the rules of $\mathcal{K}$ to β disjoint groups, where a group $\mathcal{I}_i \subseteq \mathcal{K}$ preserves a structural property on $B_i \subseteq W$ bit indices, $|B_i| \le l$, and all remaining rules are assigned to $\mathcal{D} = \mathcal{K} \setminus \cup_{i=1}^{\beta} \mathcal{I}_i$, such that $\sum_{i=1}^{\beta} |B_i| \cdot |\mathcal{I}_i| + w \cdot |\mathcal{D}|$ is minimized.*

Algorithm 5 PARMINME $(\mathcal{K}, l, \beta)$

```
1: D = K, I_0 = ∅, k = 0
2: while ∪_{s=0}^{k} I_s ≠ K and k ≤ β do
3:     k = k + 1
4:     i_k, D, I_k = MINME(D, l)
   return k, {(i_s, I_s)}_{s=1}^{k}, D
```

	Basic stats			Intersecting rules		incl. diff. actions	
	actions	rules	size, KB	rules	pairs	rules	pairs
IPv4 benchmarks							
1	3	440060	14081.9	42493	291878	10301	47530
2	185	437378	13996	59090	285157	21006	92338
3	194	410454	13134.5	45901	266329	14137	64268
4	3	410513	13136.4	45914	267105	34	38
5	208	502137	16068.3	48396	331554	14841	80511
6	101	502023	16064.7	48365	331396	58	4803
7	154	14319	458.2	912	4058	70	193
IPv6 benchmarks							
1	36	1476	188.9	288	747	228	595
2	30	1477	189.0	284	743	231	582
3	8	1475	188.8	274	732	211	457
4	20	1476	188.9	272	714	207	461

TABLE I

PROPERTIES OF ORIGINAL CLASSIFIERS.

	Original	One-stage (Fig. 2a)				Two-stage (Fig. 2b)			
	size	ingress		egress		ingress		egress	
	Kb	Rules	Size	Rules	Size	Rules	Size	Rules	Size
IPv4 benchmarks									
1	14081.9	132175	4229.6	0	0	120711	3862.7	131473	4207.1
2	13996.0	184638	5908.4	0	0	143562	4593.9	178702	5718.4
3	13134.5	146164	4677.2	0	0	123096	3939	145090	4642.8
4	13136.4	94496	3023.8	0	0	95859	3067.4	95869	3067.8
5	16068.3	176380	5644.1	0	0	137548	4401.5	174891	5596.5
6	16064.7	102281	3272.9	0	0	103166	3301.3	103727	3319.2
7	458.2	8401	268.8	0	0	7857	251.4	8441	270.1
IPv4 benchmarks with random padding to 128 bits									
1	56327.6	440060	56327.6	0	0	440060	56327.6	440060	56327.6
2	55984.3	437378	55984.3	0	0	437378	55984.3	437378	55984.2
3	52538.1	410454	52538.1	0	0	410454	52538.1	410454	52538
4	52545.6	410513	52545.6	0	0	410513	52545.6	410513	52545.5
5	64273.5	502137	64273.5	0	0	502137	64273.5	502137	64273.5
6	64258.9	502023	64258.9	0	0	502023	64258.9	502023	64258.9
7	1832.8	14319	1832.8	0	0	14319	1832.8	14319	1832.8
IPv6 benchmarks									
1	188.9	1476	188.9	0	0	1476	188.9	1476	188.9
2	189.0	1477	189.0	0	0	1477	189	1477	189
3	188.8	1475	188.8	0	0	1475	188.8	1475	188.8
4	188.9	1476	188.9	0	0	1476	188.9	1476	188.9

TABLE II

ONE-STAGE FORWARDING (FIG. 2A) AND TWO-STAGE FORWARDING (FIG. 2B).

Algorithm 5 shows the basic idea of greedy multi-group optimization with MATCH EQUIVALENCE; in a nutshell it iteratively calls to Algorithm 4 to build a group without backtracking (while there are unused groups) on the currently still unassigned rules.

Property 3. *For a classifier with N rules and w-bit headers, the time complexity of* PARMINME() *is $O\left(\beta N^2 \cdot w^3\right) + O(\beta w^2) \cdot T(\mathrm{SE})$, where $T(\mathrm{SE})$ is the time complexity of the* SE *procedure.*

Note that in practical implementations we can mitigate time complexity of the proposed algorithm in two different ways. First, we can operate on blocks of bits. Second, if the classification width is relatively large (e.g., IPv6), we can just drop bits that still preserve structural properties, not necessarily looking for maximal memory savings every time. We applied these shortcuts in our evaluation study, and did not find significant differences between these simplified versions and the full algorithms proposed in this section. In the next section, we empirically compare the efficiency of classifiers based on various structural characteristics.

V. EVALUATION

All our algorithms are applicable to multi-field classifiers with general priorities, but due to the lack of scalable real

#	Fig. 2a	Reduced classifier (Fig. 3a)							
		Group 1 stats			1 group total		All groups		
	size	W.	$\|\mathcal{I}\|$	$\|\mathcal{D}\|$	size	% of 2a	# gr.	size	% of 2a
IPv4 benchmarks									
1	4229.6	22	119681	12494	3032.7	71.7	5	2907.8	68.7
2	5908.4	22	158762	25876	4320.7	73.1	7	4062	68.7
3	4677.2	22	129651	16513	3380.7	72.3	6	3215.6	68.8
4	3023.8	22	88978	5518	2134	70.6	5	2078.9	68.8
5	5644.1	22	154498	21882	4099.1	72.6	7	3880.3	68.7
6	3272.9	22	95730	6551	2315.6	70.8	6	2250.1	68.7
7	268.8	19	6987	1414	178	66.2	6	159.6	59.4
IPv4 benchmarks with random padding to 128 bits									
1	56327.6	25	437223	2837	11293.7	20.1	3	11192.0	19.9
2	55984.3	26	436994	384	11410.9	20.4	3	11331.1	20.2
3	52538.1	25	409543	911	10355.1	19.7	3	10293.0	19.6
4	52545.6	25	409229	1284	10395.0	19.8	3	10384.6	19.8
5	64273.5	26	496548	5589	13625.6	21.2	3	13380.3	20.8
6	64258.9	26	499220	2803	13338.5	20.8	3	13178.4	20.5
7	1832.8	20	13720	599	351	19.2	3	286.3	15.6
IPv6 benchmarks									
1	188.9	15	1459	17	24	12.7	2	22.1	11.7
2	189.0	15	1461	16	23.9	12.6	2	22.1	11.7
3	188.8	16	1467	8	24.4	12.9	2	23.6	12.5
4	188.9	15	1463	13	23.6	12.5	2	22.1	11.7

TABLE III

GROUP 1 STATISTICS AND FINAL SIZES FOR ONE-STAGE INGRESS FILTER-ORDER IND. EQUIVALENT REPRESENTATION (FIG. 3A).

	Fig. 4a			Fig. 4b			Fig. 4c		
	ingress	egress	total	ingress	egress	total	ingress	egress	total
IPv4 benchmarks									
1	2643.5	2593.7	23741.7	2293.5	1549.8	19897.8	2293.5	1549.8	19897.8
2	4062	3984.7	36480.7	2871.2	3406.7	26376.3	2871.2	3406.7	26376.3
3	2923.2	2887.5	26273.1	2338.8	2620.4	21330.8	2338.8	2620.4	21330.8
4	755.9	755.9	6803.1	671	447.6	5815.6	575.1	447.6	5048.4
5	3880.3	3805.8	34848.2	2613.4	3335.4	24242.6	2613.4	3335.4	24242.6
6	1022.8	1049.9	9232.3	928.4	557.8	7985.0	928.4	557.8	7985.0
7	117.6	122.2	1063.0	109.9	110.4	989.6	102.1	110.4	927.2
IPv4 benchmarks with random padding to 128 bits									
1	11338.2	11020.3	101725.9	10103.3	11100.2	91926.6	10561.4	10103.3	94594.5
2	10726.2	10445.7	96255.3	11241.9	9854.4	99789.6	10497	9854.4	93830.4
3	10303.6	9134.1	91562.9	9876	9223.1	88231.1	9850.8	9223.1	88029.5
4	10259.2	9560.3	91633.9	10267.8	9981.8	92124.2	9852.3	9981.8	88800.2
5	12777.1	12943.4	115160.2	12774.6	11030.3	113227.1	12051.2	11030.3	107439.9
6	12348.7	12883.9	111673.5	13048.0	12112.5	116496.5	12048.5	12112.5	108500.5
7	272	273.4	2449.4	272	236.2	2412.2	257.7	236.2	2297.8
IPv6 benchmarks									
1	22.1	22	198.8	20.6	19.7	184.5	20.6	19.7	184.5
2	22.1	22	198.8	22.1	20.1	196.9	22.1	20.1	196.9
3	20.6	20.5	185.3	20.6	19	183.8	20.6	19	183.8
4	20.6	20.5	185.3	20.6	19.9	184.7	20.6	19.9	184.7

TABLE V

INGRESS, EGRESS, AND TOTAL CLASSIFIER SIZE (8XINGRESS + EGRESS) FOR NON-EQUIVALENT FIB REPRESENTATIONS.

classifiers for this case we evaluate the impact of our results on IP and IPv6 FIB classifiers. We compare the results of three different types of MATCH EQUIVALENCE with algorithms SE() (minimization of Boolean expressions) from Section IV, not only to compare results with MATCH EQUIVALENCE but also to see additional memory reductions when PARMINME() is applied to already optimized equivalent representations.

In particular, we have analyzed more than 100 real IP FIBs ranging in size from 400 kbits to 16 Mbits provided by various sources including large ISPs (unfortunately, specifics had to be omitted due to NDAs); the tables of evaluation results show only a few characteristic samples due to space constraints. Table I summarizes some basic properties of the input classifiers. The first three columns show the number of actions, number of rules, and total size. Then the table shows how often the rules in these classifiers actually intersect, i.e., how often a single header can match different rules: first how

	Fig. 2a			Fig. 2b			Fig. 3a			Fig. 3b			Fig. 3c		
	ingress	egress	total	ingress	egress	total	ingress	egress	total	ingress	egress	total	ingress	egress	total
IPv4 benchmarks															
1	4229.6	0	33836.8	3862.7	4207.1	35108.7	2907.8	2854.2	26116.6	2293.5	2258.6	20606.6	2293.5	2236.6	20584.6
2	5908.4	0	47267.2	4593.9	5718.4	42469.6	4062	3957.6	36453.6	2871.2	2836.9	25806.5	2871.2	2790.1	25759.7
3	4677.2	0	37417.6	3939	4642.8	36154.8	3215.6	3145.9	28870.7	2338.8	2305.5	21015.9	2338.8	2280.3	20990.7
4	3023.8	0	24190.4	3067.4	3067.8	27607.0	2078.9	2044.2	18675.4	671	670.7	6038.7	575.1	566.9	5167.7
5	5644.1	0	45152.8	4401.5	5596.5	40808.5	3880.3	3789.1	34831.5	2613.4	2566.4	23473.6	2613.4	2555.8	23463.0
6	3272.9	0	26183.2	3301.3	3319.2	29729.6	2250.1	2215.8	20216.6	928.4	952.7	8379.9	928.4	941.8	8369.0
7	268.8	0	2150.4	251.4	270.1	2281.3	159.6	156.6	1433.4	109.9	109.5	988.7	102.1	100.6	917.4
IPv4 benchmarks with random padding to 128 bits															
1	56327.6	0	450620.8	56327.6	56327.6	506948.4	11192.0	11190.2	100726.2	11005.6	11100.2	99145.0	10561.4	10563.3	95054.5
2	55984.3	0	447874.4	55984.3	55984.2	503858.6	11331.1	11334.4	101983.2	11241.9	11243.1	101178.3	10497	10504.4	94480.4
3	52538.1	0	420304.8	52538.1	52538	472842.8	10293.0	10294.1	92638.1	9876	9882.3	88890.3	9850.8	9851.2	88657.6
4	52545.6	0	420364.8	52545.6	52545.5	472910.3	10384.6	10386.1	93462.9	10267.8	10270.8	92413.2	9852.3	9901	88719.4
5	64273.5	0	514188.0	64273.5	64273.5	578461.5	13380.3	13378.4	120420.8	12774.6	12782.3	114979.1	12051.2	13320.1	109729.7
6	64258.9	0	514071.2	64258.9	64258.9	578330.1	13178.4	13182.2	118609.4	13048.0	13035.9	117419.9	12048.5	12149.2	108537.2
7	1832.8	0	14662.4	1832.8	1832.8	16495.2	286.3	286.8	2577.2	272	272.6	2448.6	257.7	258.5	2320.1
IPv6 benchmarks															
1	188.9	0	1511.2	188.9	188.9	1700.1	22.1	21.9	198.7	20.6	20.5	185.3	20.6	20.5	185.3
2	189	0	1512.0	189	189	1701.0	22.1	22	198.8	22.1	22	198.8	22.1	22	198.8
3	188.8	0	1510.4	188.8	188.8	1699.2	23.6	23.5	212.3	20.6	20.4	185.2	20.6	20.4	185.2
4	188.9	0	1511.2	188.9	188.9	1700.1	22.1	22	198.8	20.6	20.4	185.2	20.6	20.4	185.2

TABLE IV

INGRESS, EGRESS, AND TOTAL CLASSIFIER SIZE (8XINGRESS + EGRESS) FOR EQUIVALENT FIB REPRESENTATIONS.

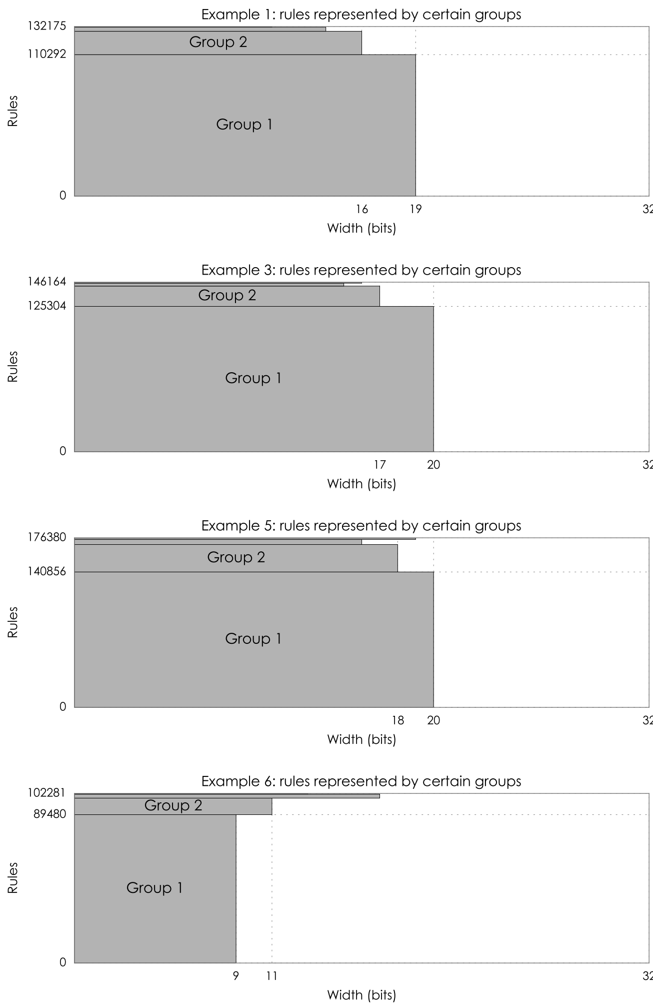


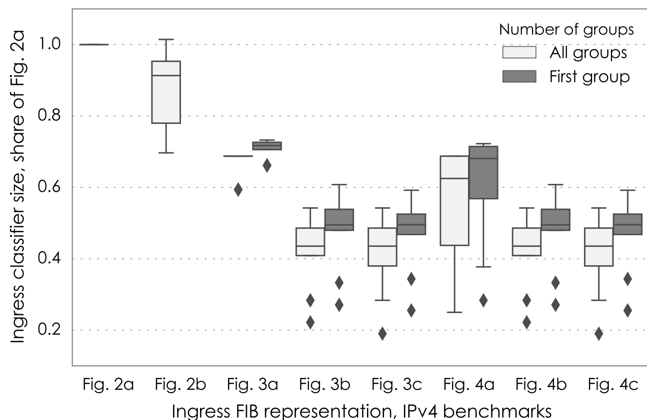
Fig. 6. Reduced classifiers shown inside original ones: four characteristic examples.

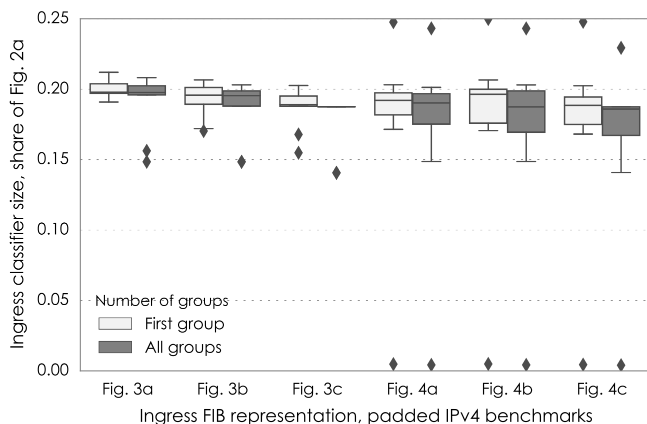
many pairs of intersecting rules there are in total and how many rules intersect with at least one other rule, and then how many of those intersections actually lead to different classification actions. Table I shows that while intersecting rules are common, they usually comprise only 10-20% of the classifiers, much less if we count only intersections that lead to different actions. This suggests a strong potential for our concepts which are based on the proposed structural properties of classifiers.

Our purpose was to show that with the proposed approach, one can actually implement IPv6 classifiers on existing IPv4 infrastructure, so we used IPv6 benchmarks generated in two different ways. The third, bottom part of every evaluation table shows our results for IPv6 FIBs from the Internet2 project [24]; again, due to space constraints we only show a few representative examples. Since infrastructural constraints have, to a certain extent, held back the advancement of the IPv6 protocol, available IPv6 benchmarks are relatively small and may not reflect the full extent to which the extra address space can be used. We have generated benchmarks that would correspond to a mature and widely used IPv6 protocol by randomly padding our IPv4 benchmarks up to the length of 128 bits. Our code for FIB reduction together with a script for reproducing the tables is available at GitHub [25].

We organize evaluation tables and figures in parallel with Figures 2, 3, and 4. Table II shows FIB sizes for one- and two-stage forwarding schemes implemented on existing distributed platforms (Fig. 2). We have applied Boolean minimization heuristics for equivalent FIB reduction on the original classifier (one-stage, Fig. 2a), original classifier with two actions corresponding to different linecards, and total size of two egress classifiers for the two LCs (two-stage, Fig. 2b). We see that equivalent heuristics can reduce IPv4 classifiers by a factor of 2-3 and ingress classifiers in two-stage forwarding by an additional 10-30%. However, there is virtually no reduction for both original IPv6 classifiers and IPv4 classifiers randomly padded to IPv6 size.

Table III dives into the details of how order-independent





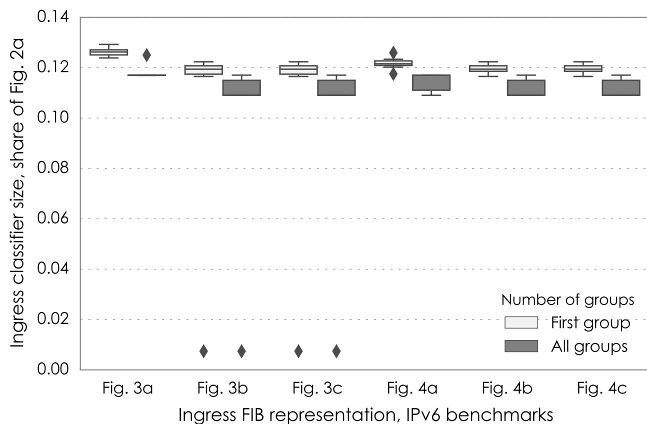


Fig. 7. Boxplots of size reduction in classifiers. Top to bottom: IPv4 benchmarks, IPv4 benchmarks padded to 128 bits, IPv6 benchmarks.

groups found with the proposed heuristics reduce the original classifier as compared to previously used techniques. Using ingress classifiers, in every case we show the width and size of the first group produced by PARMINME (these results correspond to two parallel lookups) and the total final number of groups and final size reduced by PARMINME; we also show the total size after the first group has been split and with all groups as a percentage of the one-stage forwarding table size (Fig. 2a). As we can see from Table III, on IPv4 benchmarks the PARMINME algorithm provides an additional reduction by a factor of 1.5-2 compared to the results of Boolean equivalent heuristics; in some cases, there is a big difference between classifiers with the original actions and ingress classifiers with actions for linecards, a difference that could not be exploited as well by Boolean heuristics.

Figure 6 illustrates the process of reducing a classifier graphically. For four characteristic examples from Table III, we show the number of rules along the vertical axis and width of each group along the horizontal axis. Thus, the area of the entire rectangle represents the size of the original classifier, and the shaded area shows the size of the reduced classifier; we see that in most cases we can reduce the classifier to half or less, and the vast majority of this reduction comes from the first one or two groups.

As for the IPv6 classifiers, we see in Table III that here, already in the first equivalent FIB representation, we achieve a huge advantage over both original IP and IPv6 classifiers. This is an important property of our FIB representation schemes: adding more bit identities to the original FIBs does not hinder the required memory footprint and may even help it. In both Internet2 IPv6 benchmarks and IPv4 benchmarks padded to 128 bits, we see a further reduction by a factor of 6-7 compared to representations from Table II for both ingress and egress classifiers.

Tables IV and V summarize all results; in these tables, we assume that there are 8 ingress LCs, so the total size is $8\times$ ingress size plus egress size (we remind that egress size represents the total size of FIBs on two egress LCs corresponding to different actions). We see that the total size progressively decreases from left to right, from existing two-stage forwarding schemes to equivalent FIB representations and then to non-equivalent FIB representations. Total space reductions are by a factor of 3-5 for IP FIBs and by a factor of 8-10 for IPv6 FIBs. Note also that across all benchmarks, the first group's $\mathcal{I}$ covers a large majority of the rules, and the resulting total size is comparable to the total multigroup size after applying PARMINME throughout all heuristics. This suggests that even if only two parallel lookups are available, the proposed schemes improve hugely upon existing solutions. We will revisit the problem of parallel lookups in Section VII.

The overall statistics of classifier reductions are shown in Fig. 7. There, for each FIB reduction scheme we show the data on how all benchmarks (of a certain type) are reduced in a boxplot, separately after the first group and after all groups. Fig. 7 shows that the original IPv4 classifiers are the most varied in terms of effects, but most reductions cut the size by approximately a factor of two. For padded IPv4 benchmarks we see a reduction by a factor of 5-6, and for IPv6 classifiers by a factor of 8-9; in both cases we see very stable effects across the benchmarks in our comparison.

Throughout our experiments with IPv6 benchmarks, both padded IPv4 benchmarks and real IPv6 FIBs from the Internet2 project [24], we *never* encountered a group more than 32 bits wide. The first group usually has both the most rules and the most bits; we have been able to represent all IPv6 Internet2 classifiers with two groups no more than 16 bits wide and all padded IPv4 classifiers with groups about 20 bits wide. This means that in practice, one can use existing IPv4 infrastructure designed to handle 32-bit IPv4 FIBs to

	Input			One-stage		Group 1			Tot.	Group 2			Tot.	#	Final
	A.	Rules	Size	Rules	Size	W.	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{D}\rvert$	size	W.	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{D}\rvert$	size	gr.	size
1	36	1476	188.9	1131	144.7	13	625	506	72.8	13	251	255	66.3	**6**	**59.7**
2	30	1477	189	1150	147.2	13	584	566	80	13	234	331	73.9	**6**	**65.4**
3	8	1475	188.8	1066	136.4	13	681	385	58.1	13	226	157	52.2	**5**	**48.1**
4	20	1476	188.9	1080	138.2	13	673	407	60.8	13	251	155	54.3	**5**	**50.3**

TABLE VI
SIMULATION RESULTS, IPV6 INTERNET2: MAX. 13 BIT IDENTITIES PER GROUP, FIB REPRESENTATION BASED ON NON-CONFL. RULES.

implement IPv6 classifiers. Our experiments suggest that IPv6 FIBs can be represented even on existing MPLS infrastructure, where usually MPLS lookup tables are implemented as tables with 2^{13} entries [9]. These results, for the same Internet2 IPv6 benchmarks, are shown in Table VI. Naturally, in this case there are more groups, but still 5-6 groups suffice in all cases.

Note that we do not measure explicitly lookup times and memory requirements since our approach serves as an abstract layer to find bit identities for lookups, independently of the underlying platform-dependent infrastructure implementing packet classifiers. The evaluation results confirm significant reduction in both number of entries and classification width (at least for some structural properties). In particular, the reduction of classification width can significantly reduce also lookup time for software-based approaches that are not necessary based on a single prefix (e.g., [12], [26], [27]).

VI. DYNAMIC UPDATES

Another recurring theme in classifier representations is the support of dynamic updates. Note that deletes or inserts that maintain groupwise MATCH EQUIVALENT representations are simple. If a new rule cannot be added to one of the existing groups based on l bit indices and the traditional part $\mathcal{D}$ is full, the multigroup part should be *recomputed*. The proposed update procedure is similar to the one proposed in [17, Section 7.2] to support a multigroup representation but now we require implementation of other structural properties at every group.

Example 4. Consider the following classifier, where the size of the traditional part $\mathcal{D}$ is limited to a single rule.

$$\begin{array}{cccccccc} & \#1 & \#2 & \#3 & \#4 & \#5 & \#6 & \\ R_1 & (0 & 0 & 1 & 0 & 0 & 1) & \to A_1 \\ R_2 & (0 & 0 & 0 & 1 & 0 & 0) & \to A_2 \\ R_3 & (1 & 0 & 0 & 0 & 1 & 0) & \to A_3 \\ R_4 & (1 & 1 & 0 & 0 & 0 & 1) & \to A_1 \\ R_5 & (1 & 1 & 1 & 0 & 0 & 0) & \to A_2 \\ R_6 & (1 & 1 & 0 & 1 & 1 & 0) & \to A_3 \\ R_7 & (1 & 1 & 0 & 1 & 0 & 0) & \to A_3, \end{array}$$

We first assume that the first three rules R_1, R_2, R_3 are given in step I. They can all be kept as $\mathcal{I} = \{R_1, R_2, R_3\}$ as a MATCH EQUIVALENT classifier with respect to the two bits $B = \{1, 3\}$. In step II, rule R_4 is added. It can be kept in $\mathcal{D}$, having $\mathcal{I} = \{R_1, R_2, R_3\}$, $\mathcal{D} = \{R_4\}$. Note that it cannot be simply added to $\mathcal{I}$ since the added rule have the same value in the bits of B as R_3 while their actions are different. Rule R_5 is added in step III. It can be added to $\mathcal{I}$ as $\mathcal{I} = \{R_1, R_2, R_3, R_5\}$, $\mathcal{D} = \{R_4\}$, while relying again on the same bits $B = \{1, 3\}$. The four rules in $\mathcal{I}$ have four unique values for these two bits. Next in step IV, R_6 is added. Interestingly, it can be also be added to $\mathcal{I}$ as $\mathcal{I} = \{R_1, R_2, R_3, R_5, R_6\}$, $\mathcal{D} = \{R_4\}$. This is since the rule R_6 shares its action with the rule R_3, the other rule in $\mathcal{I}$ with the same combination of these two bits. Upon the addition of R_7 in step V, it can neither be added to $\mathcal{I}$ while preserving its action-order-independence property nor to $\mathcal{D}$ due to its limited size. Accordingly, recomputation is required. By selecting new bit subsets $B' = \{5, 6\}$ we can have $\mathcal{I} = \{R_1, R_2, R_3, R_4, R_5, R_6\}$, $\mathcal{D} = \{R_7\}$ such that $\mathcal{I}$ is action-order-independent with respect to B'.

	$l = 13$			$l = 16$			$l = 24$		
	$\lvert D\rvert$=50	$\lvert D\rvert$=200	$\lvert D\rvert$=500	$\lvert D\rvert$=50	$\lvert D\rvert$=200	$\lvert D\rvert$=500	$\lvert D\rvert$=50	$\lvert D\rvert$=200	$\lvert D\rvert$=500
IPv4 benchmarks, action-order independence, $\beta = 1$									
1	6.5± 0.5	4± 0	2.8± 0.37	6± 0	3.7± 0.47	2.3± 0.47	11± 0.9	3.3± 0.47	1.3± 0.47
3	7.7± 0.47	4.7± 0.47	3± 0	7± 0.58	4± 0	3± 0	4± 0	2± 0	2± 0
4	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0
6	3± 0	1.3± 0.47	1± 0	3± 0	1.2± 0.37	1± 0	2.2± 0.37	1± 0	1± 0
7	6.7± 0.47	3± 0	2± 0	5.7± 0.47	2.8± 0.37	2± 0	3.2± 0.37	1.7± 0.47	1± 0
IPv4 benchmarks, action-order independence, $\beta = 2$									
1	5± 0	2.8± 0.37	2± 0	4± 0	2± 0	1± 0	5± 0.82	0± 0	0± 0
3	6± 0	3.5± 0.5	2.3± 0.47	4.5± 0.5	2.8± 0.37	2± 0	1.8± 0.37	0.67± 0.47	0± 0
4	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0
6	1± 0	0.17± 0.37	0± 0	1.2± 0.37	0± 0	0± 0	0± 0	0± 0	0± 0
7	5± 0.58	2.2± 0.37	1.2± 0.37	3.8± 0.37	1.7± 0.47	1± 0	0.83± 0.37	0± 0	0± 0
IPv4 benchmarks, action-order independence, $\beta = 4$									
1	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0
3	3.5± 0.5	2± 0	1± 0	2± 0	2± 0	1.8± 0.37	0.67± 0.37	0± 0	0± 0
4	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0
6	1± 0	0± 0	0± 0	1± 0	0± 0	0± 0	0± 0	0± 0	0± 0
7	3.2± 0.37	1.2± 0.37	0.17± 0.37	2.7± 0.47	0.5± 0.5	0± 0	0± 0	0± 0	0± 0
IPv4 benchmarks, non-conflicting rules, $\beta = 1$									
1	4± 0	2.5± 0.5	2± 0	4± 0	2± 0	2± 0	4.2± 0.37	2± 0	1± 0
3	4.8± 0.37	2.8± 0.37	2± 0	4± 0	3± 0	2± 0	2.7± 0.47	1.8± 0.37	1± 0
4	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0
6	2.7± 0.47	1± 0	0± 0	2± 0	1.2± 0.37	0.17± 0.37	2± 0	1± 0	0.17± 0.37
7	3.8± 0.37	2± 0	1.8± 0.37	3± 0	2± 0	1± 0	2± 0.58	1± 0	1± 0
IPv4 benchmarks, non-conflicting rules, $\beta = 2$									
1	4± 0	2± 0	2± 0	4± 0	2.2± 0.37	2± 0	4.7± 0.47	1.8± 0.37	1± 0
3	4.7± 0.47	3± 0	2± 0	4.2± 0.37	3± 0	2± 0	2.3± 0.75	2± 0	1± 0
4	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0
6	2.5± 0.5	1± 0	0.17± 0.37	2± 0	1± 0	0± 0	2± 0	1± 0	0± 0
7	3.8± 0.37	2± 0	1.7± 0.47	3.5± 0.5	2± 0	1.2± 0.37	2± 0	1± 0	1± 0
IPv4 benchmarks, non-conflicting rules, $\beta = 4$									
1	4± 0	2.5± 0.5	2± 0	4.2± 0.37	2.2± 0.37	2± 0	4.2± 0.37	1.5± 0.5	1± 0
3	4.8± 0.37	2.8± 0.37	2± 0	4± 0.58	3± 0	2± 0	2.3± 0.47	2± 0	1± 0
4	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0	0± 0
6	2.2± 0.37	1± 0	0.33± 0.47	2± 0	1± 0	0.33± 0.47	2± 0	1± 0	0.17± 0.37
7	4± 0	2± 0	2± 0	3.3± 0.47	2± 0	1.3± 0.47	2.2± 0.37	1± 0	1± 0

TABLE VII
TOTAL NUMBER OF RECOMPUTATIONS FOR DYNAMIC UPDATES, IPV4 CLASSIFIERS. MEAN AND VARIANCE SHOWN FOR 50K RANDOM SUBSETS.

Tables VII and VIII show the number of recomputations for different values of these parameters of the above heuristic when rules are added one by one; Table VII shows randomly chosen subsets of 50,000 rules from IPv4 classifiers processed in random order, with mean and variance of the number of recomputations, while Table VIII shows fully processed IPv6 classifiers. Clearly, the number of available groups β, used structural properties, the value of l, and the size of available $\mathcal{D}$ have a strong impact on required memory footprint. In particular, for IPv6 for $\beta = 4$ we never (and for IPv4, almost never) required any recomputation at all for all structural properties and all considered values of l; as well as for $\beta = 2$ and $l = 24$. The available size of traditionally represented part $\mathcal{D}$ is also important: even at a small $\mathcal{D}$ (20-50 rules for IPv6, 200-500 for IPv4), $\mathcal{D}$ has made the difference between ≈ 10 recomputations and zero recomputations for $l = 16$ and $\beta = 2$. Note also that non-conflicting rules and action-order independence lead to almost exactly the same number of recomputations for $\beta = 2$ and $\beta = 4$, while filter-order independence (shown in Table VIII) is obviously much worse; this is due to the fact that usually the number of actions in FIB classifiers is significantly smaller than the number of filters.

The usage of a traditional part $\mathcal{D}$ can be a good indicator to start recomputing the classifier in advance on background. Hence, as a practical strategy we suggest the following algo-

	$\beta = 1$									$\beta = 2$									$\beta = 4$								
	$l = 13$			$l = 16$			$l = 24$			$l = 13$			$l = 16$			$l = 24$			$l = 13$			$l = 16$			$l = 24$		
$\vert D \vert$	0	20	50	0	20	50	0	20	50	0	20	50	0	20	50	0	20	50	0	20	50	0	20	50	0	20	50
IPv6 benchmarks, filter-order independence																											
1	45	29	23	33	21	17	27	15	11	10	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	45	38	33	26	21	17	22	17	14	37	11	9	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	88	64	53	55	42	25	50	37	21	55	12	2	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	82	64	42	53	34	21	51	32	19	45	11	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
IPv6 benchmarks, action-order independence																											
1	29	18	16	20	11	10	17	8	7	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	32	26	27	21	15	13	20	15	13	20	8	1	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	34	25	24	24	16	15	21	14	13	18	1	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	29	24	24	18	13	12	17	12	9	11	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
IPv6 benchmarks, non-conflicting rules																											
1	29	18	16	20	11	10	17	8	7	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	32	26	27	21	15	13	20	15	13	20	8	1	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	34	26	24	24	16	15	21	14	13	18	1	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	29	25	23	18	13	12	17	12	9	12	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TABLE VIII
TOTAL NUMBER OF RECOMPUTATIONS FOR DYNAMIC UPDATES, IPV6 CLASSIFIERS.

rithm:

- apply the above heuristic;
- when $\mathcal{D}$ is larger than a predefined threshold (depending on system parameters and the rate of required dynamic updates), start recomputing all groups in the background;
- when computation results are ready, update the basic classifier with the newly computed groups (just switching a pointer) and empty $\mathcal{D}$, leaving there only those filters that arrived after recomputation had started.

Note that the number β of groups based on l bit indices and the size of $\mathcal{D}$ (together with structural properties of FIBs) defines the ability to absorb newly arriving prefixes and amount of recomputations. There are several ways to reduce the number of recomputations even further. First, real systems add new FIB entries in batches [9], [10], [28], with batch size derived from specific system properties and desired objectives. Second, it is relatively easy to mitigate the $O(w^3)$ factor (where w is the classification width of the original classifier) by further simplifying heuristics in Section IV or changing the bit-level resolution to block-level with multiple bits per block. The situation is more complex with $O(N^2)$ factor that can affect feasibility of representations with frequent dynamic updates. Overall, results shown in Tables VII and VIII suggest that the number of required recomputations is small in all considered scenarios except the toughest (e.g., fitting an IPv6 classifier into one group of 13 bits). Note that moving to $\beta = 4$ eliminates recomputations in almost all cases, and this setting requires only 5 pseudoparallel lookups which is practical even on constrained systems [9], [28]. Given all this, preferable deployment scenarios for our representations are wide classifiers such as IPv6 or Openflow, where moderate sizes of l with relatively small number of groups can significantly reduce or totally remove recomputations during dynamic updates.

VII. DPDK IMPLEMENTATION

In Section V we saw that multiple groups are required for FIB representations when classification width involved in the lookup process is reduced, leading to significant memory savings. On the other hand, the feasibility of multigroup implementations at linerate is still unclear. In reality we measure the number of "pseudo parallel" lookups that can be done with state of the art FIB implementation on a specific platform. For this purpose, we adopt the Intel Data Plane Development Kit (DPDK) [15] that provides LPM (LPM6) libraries with the Longest Prefix Match (LPM) rule priorities for IPv4 (IPv6) packets. The LPM implementation uses the DIR-24-8 algorithm [29] with 2 levels of serial lookups with 24 bits and 8 bits respectively. In the IPv6 case, the LPM6 implementation uses DIR-24-8-..-8 with 14 levels. We modified the original DPDK LPM implementation for two cases: single 16 bits lookup for our multi-group classification and serial 16-8-8 bits lookups for Cisco-style representation [9].

Since IPv4 implementation in DPDK uses only two consecutive lookups, we do not expect any lookup time improvement for our multi-group representation in this specific environment. In the IP case, the goal is to confirm that the desired number of parallel groups can be looked up during a single base line IP lookup. Our approach should benefit more from "instruction level parallelism" of modern CPU pipelines because the processor is not waiting for memory addresses for next lookups as in serial cases. All group addresses are known and will be efficiently prefetched by the processor. Thus, in our experiments we focus on how many 16 bits pseudo-parallel lookups can be executed in the time of a single 16-8-8 serial lookup. In the experiments, we used two physical servers with two 8 core Intel Xeon CPU E5-2670 @ 2.60GH, 96 GB DDR3 RAM, Ubuntu 14.04.2 LTS and two Intel X520 NIC with two 10GE ports. On the first server we ran the Pktgen-dpdk [30] traffic generator; on the second, our multi-group LPM implementations. Physical ports on the first NIC are connected with physical ports on the second NIC (see Fig. 8a). Pktgen generates IPv4/IPv6 packets through dedicated output port. Each LPM implementation receives packets on the input port, does lookup using the implemented strategy in the populated test FIB base, and transmits packets to the output port. In LPM code we measure the throughput (pps and bps) for IPv4 and the average number of cycles required for lookup for IPv6 (since the generator is not able to produce IPv6 packets in line rate). We use modified *rte_lpm_lookup()* calls and two *rte_rdtsc_precise()* calls before and after the lookup to get the number of cycles required for execution the set of lookups.

Figs. 8b and 8c show that it is possible to run multi-group representation on 1-6 parallel groups in the same time as a single IPv4 lookup since the throughput in these cases is above the baseline (16+8+8). This leads to significant memory size reduction with no loss in performance.

Since the baseline implementation in DPDK for IPv6 requires 14 serial lookups, we can check for improvement in lookup time. We compared multi-group representations based

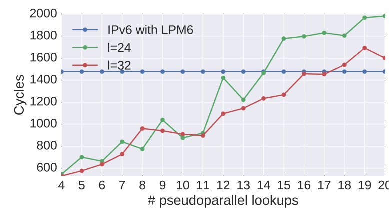
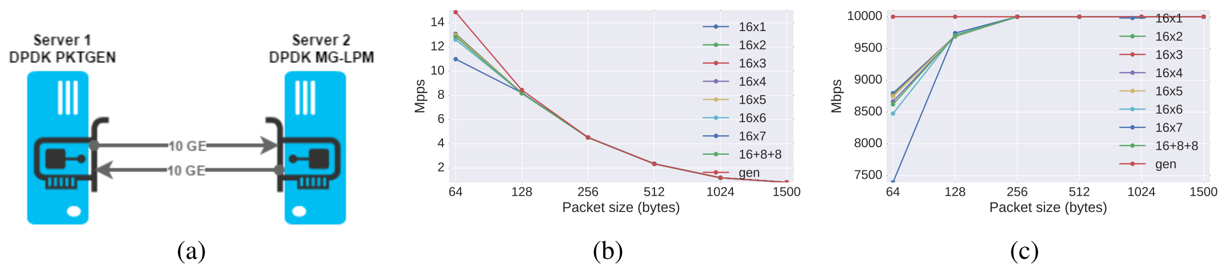


Fig. 9. Number of clock cycles as a function of the number of parallel lookups: LPM6 and multi-group representations for $l = 24$ and $l = 32$.



(a) (b) (c)

Fig. 8. Multi-group vs. 16-8-8 throughput: (a) the testbed scheme; (b) packets transmitted; (c) bits transmitted.

on $l = 24$ or 32 versus baseline DPDK implementation provided by LPM6. Fig. 9 shows that it is possible to run multi-group representation on 14-18 groups in the same time as a single IPv6 lookup implemented in DPDK. Hence, IPv6 FIBs representations that require less than this number of groups can lead to substantial improvement in the lookup time.

The source DPDK code for our experiments has been made available at GitHub [31].

VIII. Related Work

Finding efficient implementations of FIBs and packet classifiers is a well studied problem, with solutions divided into two main categories: algorithmic solutions with data structures dedicated for FIBs and TCAM-based implementations applicable to more general classifiers. The ORTC algorithm [32] obtains an optimal representation of FIBs in the minimal possible number of prefix rules. A similar approach called FIB aggregation combines rules with the same action [7], [12], [33], [34]. Interestingly, Retvari et al. [7] showed that the entropy lower bounds on the size of FIBs are much smaller than actual FIB size. Compression schemes that try to achieve these bounds have also been described [12]. Coding schemes to reduce the width of forwarding tables were suggested in [35], [36].

Several other works tried to take advantage of the structural properties of classifiers to simplify their representation and reduce their memory size. Gupta et al. [19] considered multi-field classifiers and demonstrated a common similarity between the matching requirements of the rules in each of several fields. They suggest an algorithm named RFC that categorizes the value of each field into a small number of equivalence classes and performs the matching based on the values of these classes in the different fields. These classes can be identified by a number of bits smaller than the field size. Additional compression can be achieved when the number of class combinations in different fields is small. This allows to mitigage the impact of a potential exponential increase in the memory by reducing the dependency among fields at the expense of a series of lookups that have to be performed. While this algorithm is well suited for small classifiers, its performance diminishes for larger classifiers (which are increasingly common today) or when rule updates are frequent. Similarly, Hager et al. [37], [38] consider VHDL and FPGA implementations of classifiers for which dedicated architectures have been described. Hager et al. [38] focus on longest prefix match classifiers, and design an efficient architecture with specialized circuitry. To simplify the required logic, Hager et al. [37] refers to consecutive rules with identical action as a *block*. The simplification is achieved with the observation that the internal priority between rules of the same block is not important. Then, a tree based structure of logic gates is used to determine the matching based on the results of the different blocks. A potential challenge to deal with classifiers with more than hundreds of entries is mentioned. Similarly, Wei et al. [39] suggested to preprocess the field values (e.g., changing their bit order) in order to reduce the memory size of the representation, taking advantage of its specific structure. While these works achieve the representation efficiency, their implementation in existing commodity switches with their available architectures can be challenging.

Likewise, several works [8], [26], [27], [40] suggested how to partition the multi-dimensional rule space. Hash-based solutions to match a packet to its possible matching rules have also been discussed [41] , e.g., consisting in splitting rules into subsets with disjoint rules such that a Bloom filter can be used for a fast indication whether a match can be found in each subset. Different approaches have been described to reduce TCAM space [23], [42]–[44]. They include removing redundancies [45], applying block permutations in the header space [39] and performing transformations [46]. Kogan et al. [17] introduced representations based on filter disjointness and addressed efficient time-space tradeoffs for multi-field classification, where fields are represented by ranges. Exploiting of filter disjointness for FIBs representations is considered in [47]. Recently, [48] shows how to represent multi-field classifiers on LPM infrastructures.

A recent approach suggested distributing the rules of a classifier among several limited-size TCAMs, especially in software-defined networks [20], [49], [50]. [20], [49] studies how to decompose a large forwarding table into several subtables that can be located based on previous knowledge on the paths of packets. [50] suggested to solve classification decisions that cannot be determined within a switch in special dedicated authority switches in the data plane while avoiding accessing the controller. Lossy compression was presented in [51], achieving memory efficiency by completing the classification of some small amount of traffic in a slower memory. Reducing the memory size of representing network policies by smart IP address allocation was described in [52]. Our work can serve as a level of abstraction that simplifies a classifier before it is represented with one of the above techniques. For instance, before rule splitting [53] we can reduce classifier width in order to reduce the number of disjoint sets that the rules are split into and accordingly decrease the number of

required Bloom filters. As a second example, reducing the classifier width clearly decreases the size of the binary tree that represents the classifier and decreases entropy lower bounds on the memory in the classifier's representation [7].

IX. Conclusion

Identifying appropriate invariants around which to implement lookup can significantly affect lookup efficiency and, in particular, space requirements for the corresponding data structures. In this work we have introduced new structural properties of classifiers that can be leveraged by appropriate mechanisms to achieve significantly better results than optimized equivalent solutions on distributed platforms. Our methods define an abstraction layer which does not increase lookup time and minimizes memory for lookup tables whilst remaining transparent to methods used for representation of lookup tables in network elements.

References

[1] D. Meyer, L. Zhang, and K. Fall, "Report from the IAB workshop on routing and addressing," http://www.ietf.org/internet-drafts/draft-iab-raws-report-02.txt, 2007.

[2] D. V. Krioukov, K. C. Claffy, K. R. Fall, and A. Brady, "On compact routing for the internet," *Comput. Commun. Rev.*, vol. 37, no. 3, pp. 41–52, 2007.

[3] A. Elmokashfi, A. Kvalbein, and C. Dovrolis, "BGP churn evolution: A perspective from the core," *Trans. Networking*, vol. 20, no. 2, pp. 571–584, 2012.

[4] A. Elmokashfi and A. Dhamdhere, "Revisiting BGP churn growth," *CCR*, vol. 44, no. 1, pp. 5–12, 2014.

[5] V. Srinivasan and G. Varghese, "Faster IP lookups using controlled prefix expansion," in *SIGMETRICS*, 1998.

[6] W. Eatherton, G. Varghese, and Z. Dittia, "Tree bitmap: hardware/software IP lookups with incremental updates," *Comput. Commun. Rev.*, vol. 34, no. 2, pp. 97–122, 2004.

[7] G. Rétvári, J. Tapolcai, A. Korösi, A. Majdán, and Z. Heszberger, "Compressing IP forwarding tables: towards entropy bounds and beyond," in *SIGCOMM*, 2013.

[8] T. Yang, G. Xie, Y. Li, Q. Fu, A. X. Liu, Q. Li, and L. Mathy, "Guarantee IP lookup performance with FIB explosion," in *SIGCOMM*, 2014.

[9] "Cisco XR 12000 and 12000 Series Gigabit Ethernet Line Cards," http://www.cisco.com/c/en/us/products/collateral/routers/xr12000series-router/product_data_sheet0900aecd803f856f.html.

[10] "Cisco CRS forwarding Processor Cards," http://www.cisco.com/c/en/us/products/collateral/routers/carrierrouting-system/datasheetc78730790.pdf.

[11] "CISCO 12000 Series routers," http://www.cisco.com/en/US/ products/hw/routers/ps167/index.html.

[12] A. Korösi, J. Tapolcai, B. Mihálka, G. Mészáros, and G. Rétvári, "Compressing IP forwarding tables: Realizing information-theoretical space bounds and fast lookups simultaneously," in *ICNP*, 2014.

[13] H. Song and J. Turner, "ABC: Adaptive binary cuttings for multidimensional packet classification," *Trans. Networking*, vol. 21, no. 1, pp. 98–109, 2013.

[14] K. Kogan, S. I. Nikolenko, P. Eugster, A. Shalimov, and O. Rottenstreich, "FIB efficiency in distributed platforms," in *ICNP*, 2016, pp. 1–10.

[15] "Data plane development kit (dpdk)," http://dpdk.org.

[16] "Pipelined packet switching and queuing architecture," US7643486B2, 2010.

[17] K. Kogan, S. I. Nikolenko, O. Rottenstreich, W. Culhane, and P. T. Eugster, "Exploiting order independence for scalable and expressive packet classification," *Trans. Networking*, vol. 24, no. 2, pp. 1251–1264, 2016.

[18] M. H. Overmars and A. F. van der Stappen, "Range searching and point location among fat objects," *J. Algorithms*, vol. 21, no. 3, pp. 629–656, 1996.

[19] P. Gupta and N. McKeown, "Packet classification on multiple fields," in *ACM SIGCOMM*, 1999.

[20] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the "one big switch" abstraction in software-defined networks," in *CoNEXT*, 2013.

[21] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: minimal near-optimal datacenter transport," in *SIGCOMM*, 2013.

[22] E. Allender, L. Hellerstein, P. McCabe, T. Pitassi, and M. E. Saks, "Minimizing DNF formulas and AC_d^0 circuits given a truth table," in *Conference on Computational Complexity*, 2006.

[23] K. Kogan, S. I. Nikolenko, P. T. Eugster, and E. Ruan, "Strategies for mitigating TCAM space bottlenecks," in *HOTI*, 2014.

[24] "Internet2 - forwarding information base," http://vn.grnoc.iu.edu/Internet2/fib/index.cgi.

[25] S. Nikolenko, "FIB representation code," http://github.com/snikolenko/fib-representation.

[26] B. Vamanan, G. Voskuilen, and T. N. Vijaykumar, "EffiCuts: optimizing packet classification for memory and throughput," in *SIGCOMM*, 2010.

[27] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *SIGCOMM*, 2003.

[28] "CISCO ASR 1000 series routers," http://www.cisco.com/en/US/products/ps9343/index.html.

[29] P. Gupta, B. Prabhakar, and S. P. Boyd, "Near optimal routing lookups with bounded worst case performance," in *INFOCOM*, 2000.

[30] "The pktgen dpdk application," http://pktgen.readthedocs.org.

[31] A. Shalimov, "DPDK experiments code," http://github.com/ashalimov/dpdk-mg-lpm.

[32] R. Draves, C. King, V. Srinivasan, and B. Zill, "Constructing optimal IP routing tables," in *Infocom*, 1999.

[33] Z. A. Uzmi, M. E. Nebel, A. Tariq, S. Jawad, R. Chen, A. Shaikh, J. Wang, and P. Francis, "SMALTA: practical and near-optimal FIB aggregation," in *Co-NEXT*, 2011.

[34] X. Zhao, Y. Liu, L. Wang, and B. Zhang, "On the aggregatability of router forwarding tables," in *Infocom*, 2010.

[35] O. Rottenstreich *et al.*, "Compressing forwarding tables for datacenter scalability," *JSAC*, vol. 32, no. 1, pp. 138 – 151, 2014.

[36] O. Rottenstreich, A. Berman, Y. Cassuto, and I. Keslassy, "Compression for fixed-width memories," in *ISIT*, 2013.

[37] S. Hager, F. Winkler, B. Scheuermann, and K. Reinhardt, "MPFC: massively parallel firewall circuits," in *IEEE LCN*, 2014.

[38] S. Hager, D. Bendyk, and B. Scheuermann, "Partial reconfiguration and specialized circuitry for flexible fpga-based packet processing," in *ReConFig*, 2015.

[39] R. Wei, Y. Xu, and H. J. Chao, "Block permutations in Boolean space to minimize TCAM for packet classification," in *IEEE INFOCOM*, 2012.

[40] P. Gupta and N. McKeown, "Classifying packets with hierarchical intelligent cuttings," *Micro*, vol. 20, no. 1, pp. 34–41, 2000.

[41] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor, "Longest prefix matching using bloom filters," *Trans. Networking*, vol. 14, no. 2, pp. 397–409, 2006.

[42] A. Kesselman, K. Kogan, S. Nemzer, and M. Segal, "Space and speed tradeoffs in TCAM hierarchical packet classification," *J. Comput. Syst. Sci.*, vol. 79, no. 1, pp. 111–121, 2013.

[43] K. Kogan, S. I. Nikolenko, W. Culhane, P. Eugster, and E. Ruan, "Towards efficient implementation of packet classifiers in SDN/OpenFlow," in *HotSDN*, 2013.

[44] O. Rottenstreich, I. Keslassy, A. Hassidim, H. Kaplan, and E. Porat, "Optimal in/out TCAM encodings of ranges," *Trans. Networking*, vol. 24, no. 1, pp. 555–568, 2016.

[45] A. X. Liu, C. R. Meiners, and Y. Zhou, "All-match based complete redundancy removal for packet classifiers in TCAMs," in *Infocom*, 2008.

[46] C. R. Meiners, A. X. Liu, and E. Torng, "Topological transformation approaches to TCAM-based packet classification," *Trans. Networking*, vol. 19, no. 1, pp. 237–250, 2011.

[47] S. I. Nikolenko, K. Kogan, G. Rétvári, E. R. Kovács, and A. Shalimov, "How to represent IPv6 forwarding tables on IPv4 or MPLS dataplanes," in *Infocom Workshops*, 2016.

[48] P. Chuprikov, K. Kogan, and S. I. Nikolenko, "General ternary bit strings on commodity longest-prefix-match infrastructures," in *ICNP*, 2017, pp. 1–10.

[49] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *INFOCOM*, 2013.

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

[51] O. Rottenstreich and J. Tapolcai, "Lossy compression of packet classifiers," in *ANCS*, 2015.

[52] N. Kang, O. Rottenstreich, S. G. Rao, and J. Rexford, "Alpaca: Compact network policies with attribute-carrying addresses," in *CoNext*, 2015.

[53] S. Dharmapurikar, H. Song, J. S. Turner, and J. W. Lockwood, "Fast packet classification using Bloom filters," in *ANCS*, 2006.