Exploiting Order Independence for Scalable and Expressive Packet Classification

Kirill Kogan, Sergey I. Nikolenko, Ori Rottenstreich, William Culhane, Patrick Eugster

***Abstract*—Efficient packet classification is a core concern for network services. Traditional multi-field classification approaches, in both software and ternary content-addressable memory (TCAMs), entail tradeoffs between (memory) space and (lookup) time. TCAMs cannot efficiently represent range rules, a common class of classification rules confining values of packet fields to given ranges. The exponential space growth of TCAM entries relative to the number of fields is exacerbated when multiple fields contain ranges. In this work, we present a novel approach which identifies properties of many classifiers which can be implemented in linear space and with worst-case guaranteed logarithmic time *and* allows the addition of more fields including range constraints without impacting space and time complexities. On real-life classifiers from Cisco Systems and additional classifiers from ClassBench [11] (with real parameters), 90-95% of rules are thus handled, and the other 5-10% of rules can be stored in TCAM to be processed in parallel.**

***Index Terms*—Packet classification, ternary content-addressable memory (TCAM)**

I. Introduction

Packet classification is a core functionality for popular commodity services such as Quality of Service (QoS) and access control; it has become even more prominent with the adoption of OpenFlow [38] that expresses features through hierarchical tuple matching with set actions; multi-field classification has become common. A classification rule that sets exact values for all fields can be represented by a concatenation of all fields; the classifier can then be implemented in content-addressable memory (CAM) or by a simple hash function in space linear in the number of fields. The problem becomes harder if a field is represented by a prefix (constraint on most significant bits) or a range (confining values inside an interval) since concatenation does not work. Many sophisticated software-based approaches have been proposed (cf. the comprehensive survey [57]). 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 [39], [40]. Thus, software-based approaches are either too slow or too memory-intensive even with few prefix- or range-based fields: with just 100 rules and 4 fields, N^k space is about 100MB, and $\log^{k-1} N$ time is about 350 memory accesses [40].

Ternary content-addressable memory (TCAM) was introduced to overcome performance problems for prefix matching [37]. TCAM, the *de facto* standard for classifier implementation, is a fast class of memory for matching packet headers against a set of stored entries represented by tuples of VALUE and MASK words (hiding "don't care" bits). Unlike software-based solutions, TCAM can efficiently represent multi-field classification with prefixes, but suffers an exponential blowup from range expansion; each range-based field in a rule introduces an additional multiplicative factor.

More sophisticated services have to be more expressive on existing fields or new ones, and many classifications are naturally represented by ranges. More efficient implementations for classifiers on ranges of IP or MAC addresses, dates, packet lengths, etc. are highly desired [12]. Current state-of-the-art packet classification mechanisms are limited and mostly consider five-tuples with at most two fields which include ranges, greatly restricting classification possibilities.

In this work we introduce an abstraction layer that hinges on the *order-independence* of classifiers with respect to the rules they encompass. We identify this property as essential for simple and efficient implementation of classifiers. We show that if a classifier is order-independent then new classification fields — even range-based — do not increase either time or space complexity. We use this to build efficient classification schemes. We also consider the general case with order-dependence, leading to a hybrid approach implementing order-independent rules in software with linear memory and logarithmic worst-case guaranteed lookup time, and the rest of the rules in TCAM. More precisely, our contributions are the following: (1) we identify classifier properties that guarantee zero cost for additional fields represented by ranges or prefixes transparently to the implementation scheme; (2) we propose a technique to reduce the number of classification fields represented by prefixes or ranges, yielding semantically equivalent classification results; (3) we present a technique to reduce a multi-field classification problem to a classification problem on at most two fields with guaranteed worst-case performance and linear memory, improving upon previous reduction attempts which do not provide acceptable worst-

This work is an extended version of [30].

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, St. Petersburg, Russia and Steklov Mathematical Institute at St. Petersburg, Russia; e-mail: sergey@logic.pdmi.ras.ru. O. Rottenstreich is with the department of Computer Science, Princeton University; e-mail: orir@cs.princeton.edu. W. Culhane is with Department of Computer Science, Purdue University; e-mail: wculhane@cs.purdue.edu. P. Eugster is with Department of Computer Science, Purdue University and Department of Computer Science, TU Darmstadt; e-mail: p@cs.purdue.edu.

The work of P. Eugster was partially supported by Cisco Systems and the German Research Foundation (DFG) under project MAKI ("Multi-mechanism Adaptation for the Future Internet"). The work of S. Nikolenko was supported by the Basic Research Program of the National Research University Higher School of Economics, 2015, grant No 78. We thank ACM SIGCOMM 2015 reviewers, SIGCOMM shepherd Ali Ghodsi, and Transactions on Networking Journal reviewers for their insightful comments.

case guarantees when classifiers start with more than two fields [52], [53]; (4) we demonstrate the viability of our approach through simulations on 12 classifiers from Classbench generated with real parameters [11], each with $\approx$ 50K rules on 6 fields, and on 5 real-life classifiers from Cisco Systems; we show that up to 90-95% of the rules can be implemented in linear space and with worst-case guaranteed logarithmic time, leaving only 5-10% to be implemented in TCAM.

The paper is organized as follows. Section II details the model underlying our work. Section III explains the relevance of order-independence, and Section IV identifies the main computational problems arising in our approach. Section V deals with representation of order-independent classifiers as Boolean expressions, comparing it with our approach, whereas Section VI presents heuristics to efficiently find suboptimal solutions for the computational problems. Section VII proposes new standard representations of classifiers. Section VIII presents simulation results, Section IX discusses related prior art, and Section X concludes the paper.

II. Model Description

Packets are matched by their headers according to classification rules stored in a database. A packet header contains k fields; a field i, $1 \leq i \leq k$, is a string of W_i bits. A *classifier* $\mathcal{K}$ is an ordered set of N rules $R_1 \dots R_N$. A rule R_j is an ordered set of k fields and an associated action A_j; a field F_i is represented by a range of values on W_i bits, i.e., each rule contains k ranges $R = (I_1, \dots, I_k)$, $I_i = [l_i, u_i]$. We assume the last rule of each $\mathcal{K}$ is a "catch-all" rule $R_N = (*, \dots, *)$ with $A_N =$ TRANSMIT to transmit matched packets without changes. We denote by $\mathcal{K}^{-F}$ the classifier obtained from $\mathcal{K}$ by removing a set of fields F from each rule; by $\mathcal{K}^{+F}$, the classifier obtained from $\mathcal{K}$ by extending its rules with F (with values defined separately for every rule), and simplify notation to $\mathcal{K}^{-|F|}$ and $\mathcal{K}^{+|F|}$ when the fields are clear from context.

Rules are applied sequentially, i.e., the set of rules has non-cyclic ordering $\prec$, and if a header matches both R_x and R_y for $x \prec y$, rule R_x takes precedence. Classifiers are semantically equivalent if they match the same rule for every packet.

We say that two rules R_1 and R_2 of a classifier *intersect*, or *conflict*, if there is at least one header that matches both R_1 and R_2. For instance, for $w = 4$, and $R_1 = (1\ 0\ 0\ *)$, $R_2 = (0\ 1\ *\ *)$, and $R_3 = (1\ *\ *\ *)$, R_1 and R_3 intersect (the header $(1\ 0\ 0\ 0)$ matches both rules). In contrast, R_1 and R_2 are *disjoint*. R_1 and R_2 are *order-independent* if the corresponding sets of matching headers are disjoint. For such rules, every header matches at most one of them. Two rules R_1 and R_3 are *transitively* order-dependent on a subset of fields $\mathcal{F}$ if there exists a rule R_2 such that R_1 and R_2 are order-dependent and R_2 and R_3 are order-dependent.

A classifier $\mathcal{K}$ is called *order-independent* if any two of its rules are filter-order-independent; otherwise, $\mathcal{K}$ is order-dependent. Note that order-independence is equivalent to the fact that for any classifier $\mathcal{K}'$ with the same rules as $\mathcal{K}$ sorted in a different order (except the last "catch-all" rule) any packet header p is matched by the same rule in $\mathcal{K}$ and $\mathcal{K}'$. This order-independence condition is satisfied when rules do not intersect, i.e. for each pair of rules there is at least one field in which the corresponding ranges (or prefixes) are disjoint. For instance, the classifier $\mathcal{K}$ with two rules based on two fields $R_1 = ([1,3],[4,5])$ and $R_2 = ([5,6],[4,5])$ is order-independent, but $\mathcal{K}'$ with two rules $R_3 = ([1,3],[4,5])$ and $R_4 = ([2,4],[4,5])$ is order-dependent: R_1, R_2 do not intersect since the ranges in their first field are disjoint while R_3, R_4 do intersect (e.g., $(3,4)$ matches both rules of $\mathcal{K}'$). $\mathcal{K}(S)$ denotes a classifier that uses only a subset S of fields in classification.

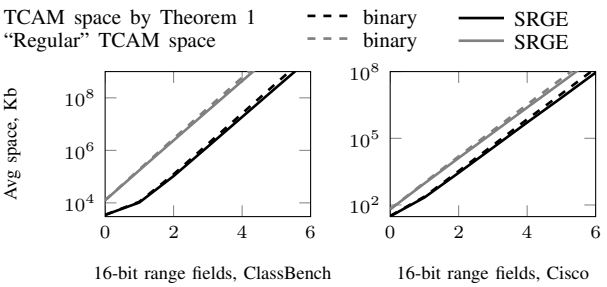


Fig. 1: TCAM space for ClassBench and CISCO classifiers as a function of the number of additional 16-bit range fields.

III. Impact of Order-independence

Order-independence of classifiers, or parts thereof, is a prevalent and powerful property for efficient classifier representation and evaluation. Classifiers with longest-prefix matches on a single field are at least order-independent among all prefixes of the same length; forwarding entries in OpenFlow [38] likely preserve order-independence for simplicity of management; many classifiers representing services with "best-match" priority schemes are order-independent. Order-dependent classifiers representing service-level-agreements (SLAs) are often unintended byproducts of uniting several services in one classifier. In our experience of over a decade of deployments, many classifiers representing services such as QoS or ACLs are order-independent in practice. Increasing the number of rules of a classifier decreases the chance that it will stay order-independent *as a whole*, but the size of its *maximal order-independent part* increases. Adding fields to each rule increases the chances of order-independence. In this section, we give a first intuition of how order-independence works and how it can drastically reduce TCAM space in a classifier.

To study the impact of order-independence, we analyze classifiers from ClassBench generated with real parameters [11] and real-life classifiers provided by Cisco Systems. The data in Table I indicate new methods for efficient representation of order-independent parts of classifiers can greatly reduce requirements for classification engines. If a classifier is not wholly order-dependent, we can lookup its order-independent and order-dependent parts separately and return the highest-priority match. An order-independent match preempts the need to match the order-dependent part. Alternatively, we can choose rules for the order-independent part which have higher priorities than any rule in the other part.

Example 1: Consider an order-independent classifier $\mathcal{K} = (R_1, R_2, R_3)$ with two fields of 5 bits each and rules $R_1 = ([1,3],[4,31])$, $R_2 = ([4,4],[2,30])$, $R_3 = ([7,9],[5,21])$. Let

	Rules		Original classifier						Classifier expanded by two 16-bit ranges					
			Original $\mathcal{K}$			By Theorem 2			Original $\mathcal{K}^{+2}$			By Theorem 1		
	Total	Order-ind., all fields	Width, bits	Space, Kb Binary	SRGE	OI wid, bits	Space, Kb Binary	SRGE	Width, bits	Space, Kb Binary	SRGE	OI wid, bits	Space, Kb Binary	SRGE
acl1	49870	49779	120	7922	7655	31	1517	1517	152	1752769	1462501	31	3857	3509
acl2	47276	44178	120	11289	11289	82	4189	4189	152	2499525	2159699	82	145925	126505
acl3	49859	47674	120	10771	10571	91	5018	5008	152	2391264	2027239	91	177771	152514
acl4	49556	46670	120	10079	9904	97	5379	5370	152	2234839	1895078	97	216994	186701
acl5	40362	38962	120	6121	6121	63	2950	2950	152	1359256	1172403	63	127818	110663
fw1	47778	43675	120	19454	19438	72	3926	3911	152	4303234	3720573	72	194313	164893
fw2	48885	48826	120	10866	10866	52	2498	2498	152	2399603	2071400	52	7382	6695
fw3	46038	41615	120	15090	15073	84	4161	4145	152	3337763	2873120	84	170688	144596
fw4	45340	42857	120	33500	33368	76	4025	4008	152	7438741	6390798	76	195130	165332
fw5	45723	39962	120	12478	12445	76	3759	3745	152	2745105	2366939	76	180420	152652
ipc1	49840	48294	120	8041	7924	50	2580	2579	152	1789100	1521153	50	52378	45391
ipc2	50000	50000	120	5859	5859	36	1757	1757	152	1301839	1123612	36	1757	1757
cisco1	584	538	120	78	78	52	34	34	152	17523	15088	52	1650	1441
cisco2	269	249	120	68	68	21	7	7	152	15662	13510	21	477	415
cisco3	95	92	120	11	11	30	3	3	152	2505	2137	30	76	71
cisco4	364	329	120	79	79	38	18	18	152	17827	15385	38	1484	1237
cisco5	148	120	120	19	19	17	5	5	152	4303	3717	17	695	590

TABLE I: Left to right: # of rules (total and in a maximal order-independent set); TCAM space under two encodings (binary [53] and SRGE [6]) for ClassBench classifiers: standard representation and reduced by Theorem 2; TCAM space for the same classifiers extended with 2 new random synthetic 16-bit range fields: standard and reduced by Theorem 1.

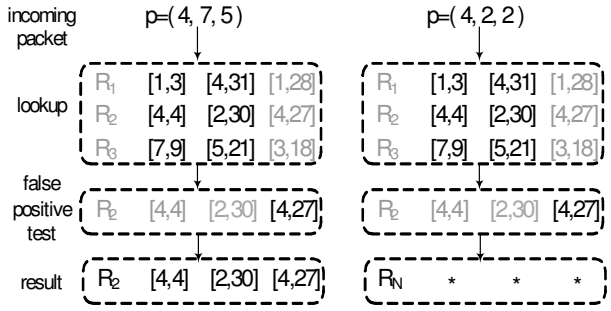


Fig. 2: Visualizing the lookup procedure on rules with a subset of fields in Example 1. Fields checked in a certain step are in black; those irrelevant for the step, in grey. Packet $(4, 2, 2)$ matches R_2 but fails the false positive check on the added field of R_2^{+1}, so the classifier returns the catch-all rule.

$\mathcal{K}^{+1}$ be a classifier that results from $\mathcal{K}$ by adding one new field of 5 bits, leading to the rules $R_1^{+1} = ([1,3],[4,31],[1,28])$, $R_2^{+1} = ([4,4],[2,30],[4,27])$, $R_3^{+1} = ([7,9],[5,21],[3,18])$.

Since $\mathcal{K}$ is order-independent, the new fields can be skipped during (the main TCAM-based) classification and a single matched rule verified to avoid false positive matches (see Figure 2). The binary encoding [53] of $\mathcal{K}^{+1}$ requires $42 + 28 + 50 = 120$ TCAM entries and the SRGE encoding [6] requires $24 + 8 + 32 = 64$ entries. On the other hand, the binary and SRGE encodings of $\mathcal{K}$ require only $6+7+10 = 23$ and $6 + 4 + 8 = 18$ TCAM entries respectively.

Theorem 1 (Fields Expansion): Let $\mathcal{K}^{+m}$ be a classifier that results from an order-independent classifier $\mathcal{K}$ by adding m new fields of any width, i.e., a rule $R = (I_1, \ldots, I_k)$ in $\mathcal{K}$ is replaced with $R^{+m} = (I_1, \ldots, I_k, \ldots, I_{k+m})$ in $\mathcal{K}^{+m}$. Then $\mathcal{K}$ with a false-positive check of a single matched rule is a semantically equivalent representation of $\mathcal{K}^{+m}$.

Proof: Adding new fields to an order-independent classifier preserves its order-independence, so $\mathcal{K}^{+m}$ is order-independent. If a header H does not match any rule of $\mathcal{K}$, H cannot match $\mathcal{K}^{+m}$ (we only add constraints). If a header H matches $R_j \in \mathcal{K}$, either H matches $R_i^{+m} \in \mathcal{K}^{+m}$ or H does not match any rule of $\mathcal{K}^{+m}$ (if a header H matches both R_i^{+m} and some R_j, $i \neq j$, it would contradict the order-independence of $\mathcal{K}^{-m}$). If $R_j \in \mathcal{K}$ is matched we check m additional fields to avoid a false-positive match. ■

By Theorem 1, introducing additional fields based on prefixes or ranges to an order-dependent classifier affects only the encoding size of its order-dependent part. The space and lookup time complexity of classification in the order-independent do not increase since new fields in the order-independent part can be ignored without affecting the classification outcome; an additional check of at most one matched rule is required to avoid a false positive match. Previously new fields based on prefixes or ranges significantly increased the complexity of software-based solutions (recall $O(\log^{k-1} N)$ lookup time in linear memory). Likewise, in TCAM-based solutions each range is converted to prefixes before configuration, and any new field based on ranges adds an additional multiplicative factor for the required TCAM space. This contributes to the fact that 5-tuple classifiers with only source and destination port fields represented by ranges are the current industry standard. Support for additional fields amenable to range rules would greatly improve classification expressiveness (e.g., ranges on dates, packet length, etc.).

The effect of Theorem 1 is shown in Table I. We add two new ranges on 16 bits each to benchmark classifiers, garnering significant space savings. We compare memory requirements which result from Theorem 1 with those of two well-known range encoding schemes: binary [53] and SRGE [6]. In addition, Figure 1 demonstrates how average required space depends on the number of range-based fields; for our approach, the exponential growth is significantly deterred. Whereas Theorem 1 shows an equivalent representation of $\mathcal{K}^{+m}$ when new fields are added, Theorem 2 states we can ignore a subset of fields if the remaining fields maintain the order-independence of the classifier. Table I shows the effect of Theorem 2: for most benchmarks, significant space savings (by a factor of 2 to 5) caused by order-independence result immediately, on the classifiers themselves rather than their extended versions. The effect is independent of range encoding schemes.

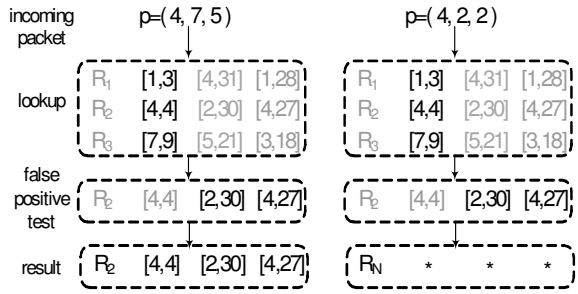


Fig. 3: Visualizing the lookup procedure on rules with a subset of fields in Example 2. Fields to be checked in a step are shown in black; irrelevant for the step, in grey. Note that packet $(4, 2, 2)$ matches $R_2^{-\{2,3\}}$ but fails the false positive check, so the classifier returns the catch-all rule.

Note that Table I does not include memory requirements for the false-positive check (on a single matched rule) since it does not require a conversion of ranges to prefixes, and a different memory (e.g., SRAM, SDRAM) can be used for its implementation which is less of a bottleneck than TCAM. Moreover the required memory for false-positive checks is only linear in the number of rules (before a conversion of ranges to prefixes) and classification width (in bits).

Example 2: Consider a classifier $\mathcal{K} = (R_1, R_2, R_3)$ with three fields of 5 bits each and rules $R_1 = ([1,3],[4,31],[1,28])$, $R_2 = ([4,4],[2,30],[4,27])$, $R_3 = ([7,9],[5,21],[3,18])$. In this case the first field is necessary but also sufficient to guarantee order independence, so we have that $\mathcal{K}^{-\{2,3\}} = \{R_1^{-\{2,3\}}, R_2^{-\{2,3\}}, R_3^{-\{2,3\}}\} = \{([1,3]),([4,4]),([7,9])\}$ is order-independent (see Figure 3). That the binary encoding of $\mathcal{K}$ requires $42 + 28 + 50 = 120$ TCAM entries and the SRGE encoding of $\mathcal{K}$ requires $24 + 8 + 32 = 64$ TCAM entries. On the other hand, the binary encoding for $\mathcal{K}^{-\{2,3\}}$ requires $2 + 1 + 1 = 4$ TCAM entries and the SRGE encoding requires only $2 + 1 + 2 = 5$ TCAM entries.

Note that removing additional classification fields of $\mathcal{K}^{-\{2,3\}}$ based on ranges can significantly reduce space complexity since the same $\mathcal{K}^{-\{2,3\}}$ is sufficient to represent $\mathcal{K}$ in Example 2. Only one false-positive check of a matched rule $R_i^{-\{2,3\}}$ is required. To further improve performance, the removed fields of R_i can be checked in parallel[1].

Theorem 2 (Fields Reduction): Let $\mathcal{K}^{-m}$ be a classifier that results from an order-independent classifier $\mathcal{K}$ by removing m fields, i.e., each $R = (I_1, \ldots, I_k)$ is replaced with $R^{-m} = (I_1, \ldots, I_{k-m})$. If $\mathcal{K}^{-m}$ is order-independent then $\mathcal{K}^{-m}$ with a false-positive check of a single matched rule is a semantically equivalent representation of $\mathcal{K}$.

Proof: If a header H does not match $\mathcal{K}^{-m}$, H cannot match $\mathcal{K}$ (we only remove new constraints). If a header H matches R_j^{-m}, H can match only R_j or H does not match $\mathcal{K}$ (if a header matched both R_i^{-m} and some R_j, $i \neq j$, it would contradict the order-independence of $\mathcal{K}^{-m}$). ■

[1]We believe that in many packet processing engines the false-positive check can be done in parallel; for example, Cisco c12000 platform has a special logic mechanism that matches up to 32 ranges in parallel at line rate [61].

By Theorem 2, detecting a false-positive match of a rule $R_i^{-m} \in \mathcal{K}^{-m}$ requires only to check m additional fields of R_i. If a false positive is found the catch-all rule is used. At most one rule can match in $\mathcal{K}^{-m}$ classifiers, which hugely impacts the complexity and feasibility of implementation.

There are significantly different complexities involved in adding and removing fields with the suggested approach. In the first case it suffices to maximize the size of the order-independent part on the same set of fields. In the second case, there is an additional tradeoff between the size of the order-independent part and the subset of fields necessary to retain order-independence. In the following section we suggest efficient representations of order-independent parts. The effect of Theorem 2 will be demonstrated in Section VIII.

IV. Efficient Representations of Classifiers

Since the tradeoff between lookup time and memory space for multi-field classification heavily depends on the number of fields k participating in classification, it is reasonable to explore ways to reduce k. For software-based solutions, any additional field that is represented by a prefix or range incurs additional complexity on the lookup time or memory space [39]. If the reduced classifier $\mathcal{K}^{-m}$ contains at most two fields, we can efficiently implement lookup in time logarithmic in N with (near-)linear memory [9], [16], [25], [53]. For TCAM-based solutions, if a classifier field is represented by a range, removing it reduces the required TCAM space proportionally to the number of prefixes used to express the range. In this section we present optimization problems as a step to achieve this goal. Moreover, removing fields can create a shorter classification format. Common TCAM formats include 72, 144, or 288 bits, so reducing the representation size across one of those barriers halves the required TCAM space. Over the course of this section, we present several ideas of how to reduce classifiers by exploiting order-independence properties, and introduce the corresponding combinatorial problems.

A. Semantically Equivalent Representations of All Rules

We begin by optimizing order-independent classifiers based on Theorem 2 (see Example 2).

Problem 1 (Fields Subset Minimization (FSM)): Find a maximal subset of fields M of an order-independent classifier $\mathcal{K}$ such that $\mathcal{K}^{-M}$ is order-independent; if there are several such subsets, choose M with maximal total width (to minimize lookup word width).

FSM can only be applied to order-independent classifiers, and it does not always reduce the number of classification fields for order-independent classifiers. It is enough that every field of a single rule is required to keep order-independence of a classifier. One way to address these shortcomings is a *multi-group* representation. A subset of classification fields may suffice for order-independence of a subset of rules of a classifier. Assume that the rules of a classifier $\mathcal{K}$ can be assigned to β groups such that: (1) each rule belongs to a single group; (2) the rules of each group are order-independent on a subset of k fields of $\mathcal{K}$ (except the "catch-all" rule);

(3) different groups can reuse the same fields to keep order-independence. By Theorem 2, a lookup to a group returns a single matched rule to be checked for false-positives on all remaining fields; the matched non-false-positive rule with the highest priority (from exactly β matched rules) is returned. We call this the *multi-group* representation of $\mathcal{K}$. Multi-group representations exist for any order-dependent classifier, but each group should be order-independent. The following theorem immediately follows by construction.

Theorem 3: The multi-group representation of a classifier is semantically equivalent to the original classifier.

Lookups to groups can be issued in parallel. If each group in a multi-group representation can be order-independent on at most two fields, we have a semantically equivalent SW-based representation of a classifier in linear memory and with worst-case guaranteed logarithmic lookup time. Example 3 shows how the rules of a classifier $\mathcal{K}$ can be split into groups.

Example 3: Consider $\mathcal{K} = \{R_1, R_2, R_3, R_4, R_5\}$ with three fields of four bits each and rules $R_1 = ([5, 10], [4, 7], [4, 5])$, $R_2 = ([1, 4], [4, 7], [4, 5])$, $R_3 = ([1, 9], [1, 3], [4, 6])$, $R_4 = ([1, 9], [4, 7], [1, 3])$, $R_5 = ([1, 9], [4, 7], [5, 6])$. While the set of all rules $\{R_1, R_2, R_3, R_4, R_5\}$ is not order independent (for instance, $R_1 \cap R_5 \neq \emptyset$), the rules can be divided into two independent subsets, $\{R_1, R_2, R_3\}$ and $\{R_4, R_5\}$. The first two fields are necessary to keep the first group order-independent and the last field is enough to keep the second group order-independent.

Figure 4 shows a visualization of the lookup procedure after the grouping has been accomplished: for an incoming packet $p = (2, 4, 5)$, we issue parallel lookups for the two groups; these lookups return R_2 and R_5 and look only at fields $\{1, 2\}$ and $\{3\}$ respectively. Then we check if it was a false positive, matching p against the rest of the fields for the returned rules, and then select the match by rule priority. At most one rule will make it to the false positive test for each group. In case when no rule matches or the final test finds that the result was a false positive, the catch-all rule is sent to the priority comparison, so each group sends exactly one rule to the priority comparison for any packet.

In some order-independent classifiers removing any of the fields make it be order-dependent and accordingly the FSM cannot reduce the number of fields. This leads us to the following optimization problem that can be applied also to order-dependent classifiers.

Problem 2 (l-Groups of Rules (l-MGR)): Given a classifier $\mathcal{K}$, find an assignment of $\mathcal{K}$'s rules to a minimal number of disjoint groups such that different groups of rules can be based on the same or different subset of at most $l \leq k$ fields, and each group is order-independent on these fields.

B. Representation of a Subset of Rules

Problem 2 shows how to deal with order-dependent classifiers. However, the number of lookups that can be issued in parallel is a system parameter, and the minimal number of groups found by solving Problem 2 may be unacceptable. In this situation, we can decompose an order-dependent classifier $\mathcal{K}$ into order-independent part $\mathcal{I}$ and order-dependent part $\mathcal{D}$.

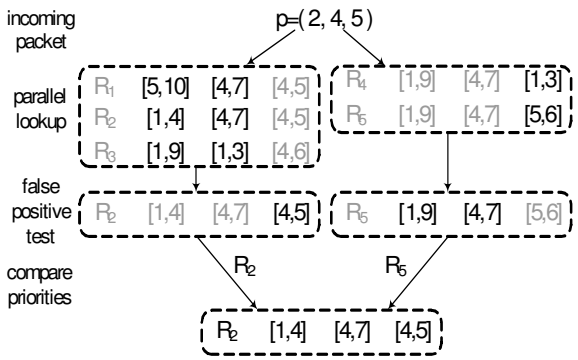


Fig. 4: A visualization of the lookup procedure with a multi-group representation of the rules in Example 3. Fields to be checked on a certain step are in black; those that are irrelevant for this step, in grey.

We can perform parallel lookups on both parts and return the best matched rule. TCAM or another general implementation can be used to store $\mathcal{D}$. Since in practice a subset of order-dependent rules on k fields should be significantly smaller than N, significant TCAM space can be saved. The problem is finding which rules to put in I. One possibility is to find a maximal order-independent subset of rules on all k fields and run Problems 1 or 2 on $\mathcal{I}$. Finding maximal order-independent set on k fields is interesting when we extend already existing classifiers with new fields (Theorem 1). Another direction is finding an order-independent part that already has the desired properties.

Problem 3 (Maximum Rules Coverage (l-MRC)): Given a classifier $\mathcal{K}$ and a positive number $l \leq k$, find a maximal subset $\mathcal{I} \subseteq \mathcal{K}$ which is order-independent on at most l fields.

Example 4: Consider a classifier $\mathcal{K} = \{R_1, R_2, R_3\}$ with three fields and the first three rules from Example 3: $R_1 = ([5, 10], [4, 7], [4, 5])$, $R_2 = ([1, 4], [4, 7], [4, 5])$, $R_3 = ([1, 9], [1, 3], [4, 6])$. If $l = 2$, we can have a maximal subset $\mathcal{K}' = \mathcal{K}$ of three rules, as the first two fields guarantee that the set $\mathcal{K}$ of these three rules is order-independent. Likewise, if $l = 1$ and only one field can be selected to distinguish between the rules, we can have order-independent sets of two rules. By considering the first field, we can have the subset $\mathcal{K}' = (R_1, R_2)$. Using only the second field we can have other possible order-independent subsets such as $\mathcal{K}' = (R_1, R_3)$ or $\mathcal{K}' = (R_2, R_3)$. We can also see that the three rules together are not order-independent based on any single field and accordingly the three mentioned possible subsets are all maximal subsets.

The generalized version of l-MRC problem reuses the advantages of the decomposition into I and D and multi-groups representation.

Problem 4 ((β, l)-MRC): Given a classifier $\mathcal{K}$ and two positive numbers $l \leq k$ and β, find a maximal subset of rules $\mathcal{I} \subseteq \mathcal{K}$ that can be assigned to at most β groups, where each group is order-independent on at most l fields.

In some cases, it may be feasible to send a few more rules to $\mathcal{D}$ and thus significantly reduce the number of groups; this usually happens in practice with more general rules at the bottom of a classifier.

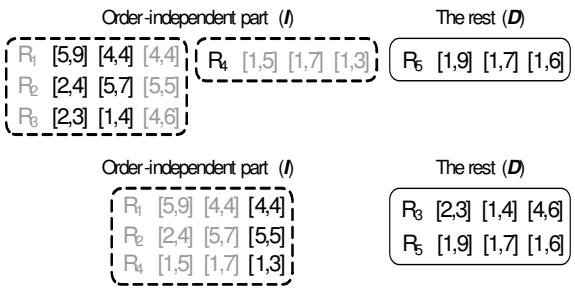


Fig. 5: Example 5. Top: a multi-group representation of $\mathcal{I} = \{R_1, R_2, R_3, R_4\}$ and $\mathcal{D} = \{R_5\}$. Bottom: a more compact representation of $\mathcal{I} = \{R_1, R_2, R_4\}$ and $\mathcal{D} = \{R_3, R_5\}$. Fields to be checked in a group lookup are in black; those that are irrelevant for this step, in grey.

Example 5: Consider a classifier $\mathcal{K}$ with three fields of five bits each and five rules $R_1 = ([5,9],[4,4],[4,4])$, $R_2 = ([2,4],[5,7],[5,5])$, $R_3 = ([2,3],[1,4],[4,6])$, $R_4 = ([1,5],[1,7],[1,3])$, $R_5 = ([1,9],[1,7],[1,6])$. The maximal order-independent subset on all three fields is $\mathcal{I} = \{R_1, R_2, R_3, R_4\}$, and one possible multi-group representation of $\mathcal{I}$ consists of two groups: $\{R_1, R_2, R_3\}$ (based on the first two fields) and $\{R_4\}$. But if we set $\mathcal{D} = \{R_3, R_5\}$, the rest will form a single group which is order-independent on a single field, the third one; see illustration on Fig. 5.

C. Efficient Cache Implementation

The (β, l)-MRC problem maximizes the number of rules in $\mathcal{I}$ that can be assigned to at most β groups that are order-independent on at most l fields. Previously, we discussed a decomposition of $\mathcal{K}$ into $\mathcal{I}$ and $\mathcal{D}$, returning the best match between them. Now we wish to construct $\mathcal{I}$ in such a way that if a non-catch-all rule of $\mathcal{I}$ is matched, the lookup to $\mathcal{D}$ is no longer required. Informally, if such $\mathcal{I}$ exists, its content can be configured in cache; we just need to guarantee that there exist no two rules $R_1 \in \mathcal{I}$ and $R_2 \in \mathcal{D}$ such that R_1 "intersects" with R_2 with priority $R_2 \prec R_1$. As a result, we formulate the following variant of the (β, l)-MRC problem.

Problem 5 ((β, l)-MRCC): Given a classifier $\mathcal{K}$ and two positive numbers $l \leq k$ and β, find a maximal subset of rules $\mathcal{I} \subseteq \mathcal{K}$ that can be assigned to at most β groups, where each group is order-independent on at most l fields and no two rules $R_1 \in \mathcal{I}$ and $R_2 \in \mathcal{D}$ such that R_1 "intersects" with R_2 and $R_2 \prec R_1$.

The built $\mathcal{I}$ by (β, l)-MRCC not only has a space-efficient representation but also implements a power-efficient solution that does not need lookups to $\mathcal{D}$ if $\mathcal{I}$ is matched; this is important because TCAMs can suffer from power-inefficiency.

D. Resolution of Optimization

In case when only a part of the fields are represented by ranges or ranges are remapped to ternary bit-strings the boundaries of real classification fields as SrcIP, DstIP, ToS, etc., are removed. A rule becomes a ternary bitstring where any number of bits can be grouped to *virtual* fields.

Example 6: Consider a classifier $\mathcal{K}$ with two fields of four bits each and four rules: $R_1 = (\texttt{100}\star,\ \texttt{001}\star)$, $R_2 = (\texttt{1010},\ \texttt{0001})$, $R_3 = (\texttt{000}\star,\ \star\star\star\star)$, $R_4 = (\texttt{001}\star,\ \star\star\star\star)$. In this case, the first field of four bits is enough for order-independence, so if we run FSM on two four-bit fields, we get $R_1^{-1} = (\texttt{100}\star)$, $R_2^{-1} = (\texttt{1010})$, $R_3^{-1} = (\texttt{000}\star)$, $R_4^{-1} = (\texttt{001}\star)$. However, note that only the first and third bits suffice to provide order-independence in this case, so if we treat $\mathcal{K}$ as 8 one-bit fields, FSM yields a classifier of width two: $R_1^{-6} = (10)$, $R_2^{-6} = (11)$, $R_3^{-6} = (00)$, $R_4^{-6} = (01)$.

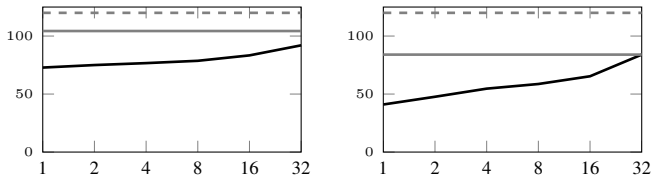


Fig. 6: Classifier width (in bits) as a function of virtual field width for ClassBench and CISCO classifiers. Dashed grey line presents average original classifier width; grey solid line shows average classifier width after MinDNF reduction (see Section V); black line illustrates classifier width after FSM optimization.

We have performed experiments on how the resulting reduced classifier width depends on the width of virtual fields. The results are summarized in Figure 6 that shows average widths of ClassBench and CISCO classifiers for different resolutions.

In light of Example 6, solutions to problems in this section are very promising even for the case of a single field represented by a range. This is actually the case of minimization of representation in forwarding tables. Since every prefix corresponds to a single range, finding a maximal order-independent set of ranges is equivalent to maximum independent sets in interval graphs, i.e. a graph in which the nodes are 1-dimensional intervals (e.g., time intervals) and there is an edge between two intervals iff they intersect. This problem can be solved exactly in $O(N \log N)$ time using the earliest-deadline-first (EDF) algorithm [26].

Using the order-independent property of a classifier can in some sense reduce its representation memory size beyond the theoretical entropy lower bounds presented in [41]. Of course, this relates to the space required in the first step of the classification and does not include the additional memory required to examine the case of a false-positive match.

Consider for instance, a classifier with four rules R_1, R_2, R_3, R_4 defined on a single field of $W = 8$ bits such that $R_1 = [148, 148] = (10010100), R_2 = [83, 83] = (01010011), R_3 = [165, 165] = (10100101), R_4 = [102, 102] = (01100110)$. The rules are associated with four distinct actions A_1, A_2, A_3 and A_4. We can see that two bits are enough in order to distinguish between the rules. These can be for instance the third and the seventh bits which have the values of $00, 01, 10$ and 11 in the four rules R_1, R_2, R_3, R_4, respectively. Accordingly, the representation of this subset of the bits of the rules together with the encoding of the action of each rule (in two additional bits) has a total space of $4 \cdot (2 + 2) = 16$ bits.

For this classifier, the binary tree representation T includes 4 leaves and a total of $27 < (32 = 4{\cdot}W)$ nodes since some nodes for the first bits in the rules are shared among more than one rule. The suggested *XBW-l* transform [41] for the tree, denoted xbwl(T), consists of three strings (S_{last}, S_I, S_α), where the bit length of the binary strings S_{last}, S_I that encodes tree structure equals the number of nodes. Likewise, the length of string S_α that encodes the actions equals the number of leaves times the size of each action in bits. This results in a space requirement of $27+27+4\cdot 2 = 62$ bits, almost four times the space required in the suggested scheme. Furthermore, the DAG representation [41], a more complicated encoding scheme for the tree T, cannot reduce the space required since in this example actions of the rules are disjoint and the tree does not have any isomorphic subtrees. In future work, we plan to compare the size of forwarding table representations computed by heuristics proposed in [41] with the results of semantically equivalent representations considered in this section. We believe that results for representations of IPv6 forwarding tables should be even better since in wider classifiers there is a better chance to find more order-independent rules on fewer bits.

V. Representing Classifiers as Boolean Expressions

In this section, we discuss a special case when each field is represented by a prefix, so fields can be represented by a string of individual bits. This special case lets us represent classifiers as Boolean expressions and apply well known Boolean optimization techniques to reduce the width and number of rules. For this exposition, we concatenate all fields and treat a rule as a ternary bit-string in the Value–Mask–Action format (since fields have a fixed width, we do not lose information with this transformation). In this part we consider order-independent classifiers; we decide whether an action should be applied to a packet, i.e., compute a Boolean function.

A rule $s = s_1 s_2 \dots s_k$, $s_j \in \{0, 1, *\}$, can be expressed as a conjunction $f_s(x_1, \dots, x_k) = \bigwedge_{s_i=1} x_i \wedge \bigwedge_{s_i=0} \bar{x}_i$. Further, an order-independent set of rules $\mathcal{R} = \{R_1, \dots, R_N\}$ with rules $s_1, \dots, s_k$ can be written as an unordered disjunction of individual rules, i.e., as a formula of depth 2 in disjunctive normal form (DNF, disjunction of conjunctions).

Example 7: The following set of rules with width 5

$$\begin{array}{lllll} (0 & 1 & * & 0 & *) \\ (* & 1 & 0 & 1 & *) \\ (* & 1 & 1 & 1 & 0) \\ (* & 1 & 1 & 1 & 1) \end{array} \quad \begin{array}{l} \to 1 \\ \to 1 \\ \to 1 \\ \to 1 \end{array}$$

is order-independent and equivalent to $f(x_1, x_2, x_3, x_4, x_5) = (\bar{x}_1 \wedge x_2 \wedge \bar{x}_4) \vee (x_2 \wedge x_4 \wedge \bar{x}_3) \vee (x_2 \wedge x_3 \wedge x_4 \wedge \bar{x}_5) \vee (x_2 \wedge x_3 \wedge x_4 \wedge x_5)$.

Thus, minimizing memory for representation of order-independent rules is equivalent to the following problem.

Problem 6 (MinDNF): For a given Boolean function f, find a minimal size DNF representation for f.

DNF minimization may result in both reducing the number of rules (merging clauses in the DNF) and reducing the width of lookup necessary for classification (removing extra variables in the clauses).

Example 8: Let us minimize the set of rules from Example 7. We apply the well-known resolution heuristic [42] to clauses in the DNF representation of f:

$$\begin{array}{rcl} (x_1 \wedge \bar{x}_2 \wedge \bar{x}_5) \vee (x_1 \wedge \bar{x}_2 \wedge x_5) & = & (x_1 \wedge \bar{x}_2), \\ (x_1 \wedge \bar{x}_2) \vee (x_1 \wedge \bar{x}_2) & = & x_2. \end{array}$$

At this point, we have $f(x_1, x_2, x_3, x_4, x_5) = (\bar{x}_1 \wedge x_2) \vee x_2$, and we can subsume the first clause into the second: $f(x_1, x_2, x_3, x_4, x_5) = x_2$. After minimizing the corresponding DNF expression, we can express the result as a set of rules, a single rule in this case: $(\star 1 \star\star\star) \to 1$. Thus, in this example we have reduced the number of rules from four to one and have reduced the number of bits participating in the lookup from four (to apply the rules in Example 7 as stated, we would have to query all bits except bit 4) to one (bit 2).

The *MinDNF* problem has been extensively studied in complexity theory. Its decision version (given a formula φ in DNF and a number k, find whether there exists an equivalent DNF with size less than k) is unlikely to be in NP. It is obviously in $\mathrm{NP}^{\mathrm{NP}}$: to solve this problem, one needs to guess a small DNF and verify with the NP oracle that solves the satisfiability problem that these two formulas are indeed equivalent. In fact, *MinDNF* has been shown to be $\mathrm{NP}^{\mathrm{NP}}$-complete ($\Sigma_2^P$-complete), and some inapproximability results have also been proven [24], [54]. In practice, classical heuristics for *MinDNF* were based on Karnaugh maps and the Quine–McCluskey algorithm.

A different view of the problem appeared for the case of functions given by truth tables [1]; this means that the input size is exponential in the number of variables, and an algorithm is thus also allowed to use exponential memory. In this case, one can represent *MinDNF* as a special case of the SetCover problem which leads to using the Greedy SetCover algorithm to find minimal DNFs (see Algorithm 3). This algorithm has been shown to be $O(n)$-approximate, where n is the number of variables in the formula (for functions given by truth tables, $O(\log T)$-approximate, where T is the truth table size), and a matching lower bound has also been provided [1]. However, this approach is impractical for classifier optimization since practical classifiers look up several hundred bits, and it would be infeasible to construct the truth table explicitly.

At the first glance *MinDNF* is a more general problem than *FSM* since it can reduce both classification width and the number of rules. However, due to an additional check for a false-positive match *FSM* can significantly reduce classification width beyond optimal results of *MinDNF*.

Example 9: Consider the classifier from Example 6: $R_1 = (100\star,\ 001\star)$, $R_2 = (1010,\ 0001)$, $R_3 = (000\star,\ \star\star\star\star)$, $R_4 = (001\star,\ \star\star\star\star)$. As we have discussed in Example 6, FSM can reduce the width of $\mathcal{K}$ from eight bits to four (if we treat fields as indivisible) or even two (if we are able to get down to the bit level). On the other hand, the only MinDNF heuristic applicable here is the resolution rule that can be applied to R_3 and R_4, getting $R_1 = (100\star,\ 001\star)$, $R_2 = (1010,\ 0001)$, $R_3' = (00\star\star,\ \star\star\star\star)$, This classifier has width 8; if we discard bits with identical values (second bit in the first field), we get width 7, but the width savings are still small compared to FSM.

	Original classifier					MinDNF reduced						OI
	Rules, orig.	Rules, OI	Rules, binary	Rules, SRGE	Wid, bits	Rules, binary	Wid, bits	Red., bits	Rules, SRGE	Wid, bits	Red., bits	wid, bits
acl1	49870	49779	67511	65240	120	67505	90	90	65234	90	90	31
acl2	47276	44178	90772	90772	120	90233	104	104	90230	104	104	82
acl3	49859	47674	85252	83630	120	85226	106	106	83605	106	106	91
acl4	49556	46670	77837	76424	120	77755	106	106	76338	106	106	97
acl5	40362	38962	47514	47514	120	46261	96	94	46249	96	94	63
fw1	47778	43675	159525	159519	120	159461	112	112	159458	112	112	72
fw2	48885	48826	92646	92646	120	92316	88	88	92316	88	88	52
fw3	46038	41615	122495	122477	120	122259	112	112	122244	112	112	84
fw4	45340	42857	278887	277921	120	277799	104	104	276807	104	104	76
fw5	45723	39962	99574	99412	120	99421	112	112	99273	112	112	76
ipc1	49840	48294	66718	65734	120	66715	112	112	65731	112	112	50
ipc2	50000	50000	50000	50000	120	50000	112	112	50000	112	112	36
cisco1	584	538	603	603	120	475	104	86	474	104	86	52
cisco2	269	249	565	565	120	564	104	84	564	104	84	21
cisco3	95	92	92	92	120	92	88	68	92	88	68	30
cisco4	364	329	629	629	120	629	104	84	629	104	84	38
cisco5	148	120	139	139	120	139	104	84	139	104	84	17

TABLE II: Experimental results of MinDNF reduction in the order-independent subsets of classifiers.

Table II shows how MinDNF heuristics apply to our experimental set of classifiers; we have applied MinDNF to the order-independent set from Table I. The number of binary rules is sometimes significantly larger than the number of original rules, and MinDNF does not have a significant effect on width, i.e., few bits become purely "don't care" bits even after MinDNF-style reductions. In Table II, we show both "pure" width (cutting out bits that are purely "don't care") and further reduced width where we have cut out bits that have the same value, not necessarily "don't care". This is a Boolean counterpart of our reduction shown in Section III since we can use a single look up to check bits that always have the same value, and they do not change which rule matches an incoming packet. MinDNF does not significantly decrease width even with this reduction.

The example of FSM shows us that algorithms with usage of an additional constant number of checks for false-positive match can significantly reduce a required space of an optimal representation of Boolean expressions. We believe that developing new Boolean expression minimization techniques that are based on the results of Theorem 2 is a very promising direction.

VI. Proprietary Heuristics

The reduction in classifier size with a false-positive check opens new horizons that are not addressed by general Boolean expression minimization. In this section we present algorithms for problems from Section IV and study their complexity.

A. Exact Algorithms

We first consider the existence of algorithms to build exactly the proposed semantically equivalent representations of classifiers.

FSM is NP-complete by reduction from the SetCover problem. Since FSM is exponential on k, an exact algorithm is feasible only for a sufficiently small number of fields (e.g., the 5-tuple case).

Algorithm 1 checks order-independence of a classifier with N rules and a set M of m removed fields with complexity $O(N^2(k-m))$. To solve FSM, we can use a binary search.

Algorithm 1 IsOrderIndependent(M)

```
for i = 1 → N − 1 do
    for j = i + 1 to N do
        if R_i^{−M} ∩ R_j^{−M} ≠ ∅ then return False
return True
```

Algorithm 2 FSMBinSearch(k, min, max)

```
m = ⌊(min+max)/2⌋
if min = max then return m
for M ⊆ {1, ..., k}, |M| = m do
    if IsOrderIndependent(M) then
        min = m
        return FSMBinSearch(k, min, max)
max = m − 1
return FSMBinSearch(k, min, max)
```

If the classifier is order-independent for some removed subset with $\frac{k}{2}$ fields, try to find a subset of size $\frac{3k}{4}$, otherwise check order-independence for subsets of size $\frac{k}{4}$, and so on. We call this algorithm FSMBinSearch. At worst we need to run IsOrderIndependent once for each subset of size $\frac{k}{2}$, once for each subset of size $\frac{3k}{4}$ and so on, $\binom{k}{k/2}+\binom{k}{k/4}+\ldots+\binom{k}{1} < 2^{k-1}$ times in total. Thus, we get the following theorem.

Theorem 4: FSMBinSearch$(k, 0, k-1)$ finds a minimal subset of k fields for which an original classifier is order-independent in time $O(k2^{k-1}N^2)$.

When the number of fields grows (e.g, by a resolution increase as in Section IV-D or adding new fields), FSM requires approximation heuristics.

The 2-MRC problem is NP-complete already for the case when $k = 2$ [7]. Since the (β, l)-MRC problem simply generalizes the 2-MRC problem when $\beta = 1$, $k = 2$, we can directly deduce based on the hardness of the 2-MRC problem that the (β, l)-MRC is NP-complete as well. As considered in Section IV-D for $k = 1$, only the 1-MRC problem has an exact solution by EDF algorithm in $O(NlogN)$ time [26]. Unfortunately, even the 2-MRC problem is exponential on N, so we do not consider exact algorithms similar to FSM when a number of fields is small.

B. Approximate Solutions

Algorithms for the FSM problem are exponential in the number of fields k; exact algorithms for the other considered problems are exponential even in the number of rules N. Therefore, as k and N grow, approximations are needed. In this section, we discuss efficient approximate algorithms that are crucial to achieve space savings.

1) Representation of All Rules:

Theorem 5: FSM is reducible to SetCover in $O(k \cdot N^2)$ time and has an approximation factor of $2 \cdot \ln(N) + 1$.

Proof: Given an instance of the FSM problem, i.e., a set of N rules $R_1, ..., R_N$, each with k fields, we would like to find a subset of the fields that can distinguish between any pair of rules. A field distinguishes between two rules if they do not intersect in the field. We look at the (unordered) possible pairs

Algorithm 3 Algorithm GreedySetCover

1: $\mathcal{X} \leftarrow \emptyset$; $\mathcal{T} \leftarrow \mathcal{S}$
2: **while** $\mathcal{T}$ not empty **do**
3: Choose S_i in $\mathcal{T}$ that contains maximal number of uncovered elements in $\mathcal{U}$
4: $\mathcal{X} \leftarrow \mathcal{X} \cup \{S_i\}$
5: $\mathcal{T} \leftarrow \mathcal{T} \setminus S_i$
6: **return** $\mathcal{X}$

of rules as the universe U that we would like to cover, i.e., to make sure that each of these pairs of rules can be distinguished by one of the fields. We define $U = \{(i,j)|i < j, i, j \in [1, N]\}$. The size of U is $|U| = \binom{N}{2} \leq 1/2 \cdot N^2$. We also define k sets $S_1, ..., S_k$ that are used to cover U such that the set S_ℓ represents the ℓ^{th} field out of k. For $\ell \in [1, k]$, we define $S_\ell = \{(i,j)|i < j, i, j \in [1, N], R_i(\ell) \cap R_j(\ell) = \emptyset\}$. S_ℓ contains all pairs of rules that do not intersect in this field. To construct $S_1, ..., S_k$, one has to check all pairs out of N rules. Therefore, this construction requires a time complexity of $O(k \cdot N^2)$ – polynomial in size of the input to the FSM problem. Given a solution to the set cover instance, i.e., a subset of $S_1, ..., S_k$, the corresponding set of fields are guaranteed to distinguish between any pair of rules and therefore comprise an admissible solution to the FSM problem.

Using Algorithm 3, in which each step selects an additional set with maximal number of uncovered elements, we can achieve an approximate solution to the FSM problem. In such a solution, the number of selected fields is within a constant factor of the optimal number of fields. The approximation ratio is $H(|U|) = H(\binom{N}{2})$, where H is the harmonic number, such that the approximation ratio satisfies $H(|U|) \leq \ln(|U|) + 1 \leq 2 \cdot \ln(N) + 1$. ∎

Next we consider an estimate for a lower bound on β (number of groups) in multi-group representations and how that bound relies on N and k for order-independent classifiers.

Theorem 6: 1. For each N and $k \geq 2$, there exists an order-independent classifier $\mathcal{K}$ of N rules on k fields such that it cannot be divided into less than $\sqrt{N} - O(1/\sqrt{N})$ groups where each group is order-independent by a single field.

2. Moreover, if $k \geq 4$, there exists $\mathcal{K}$ such that it cannot be divided into less than $\sqrt{N} - O(1/\sqrt{N})$ groups where each group is order-independent by two fields.

3. Moreover, if $N \geq 2^k$, there exists $\mathcal{K}$ such that it cannot be divided into less than 2^{k-l} groups where each group is order-independent by some l fields.

Proof: 1. For some natural n, consider a classifier on two fields with $n(n-1)$ rules; the rules span all pairs of intervals $([i,i],[j,j])$ for $1 \leq i \neq j \leq n$. Any group of rules that are independent with respect to any single field cannot contain more than n rules (since there are only n distinct intervals in each field), so there are at least $n-1$ groups.

2. To extend this counterexample to pairs of fields, consider a classifier on 4 fields that contains $n(n-1)(n-2)(n-3)$ rules that implement all quadruples of intervals $([i,i],[j,j],[k,k],[l,l])$, $1 \leq i \neq j \neq k \neq l \leq n$. Now any group of rules that are independent with respect to any pair of rules can contain at most $n(n-1)$ rules, so there are at least $(n-2)(n-3)$ groups in total.

3. Now the example is a classifier consisting of 2^k rules that implement all possible combinations of two disjoint intervals (say, $[1,1]$ and $[2,2]$) in k rules. Now any group independent with respect to l fields can contain at most 2^l rules (exhausting all possible combinations of $[1,1]$ and $[2,2]$ of length l), so there must be at least 2^{k-l} such groups. ∎

Theorem 6 shows that a number of groups depends on $\sqrt{N}$ or is exponential on k in the worst case. Can we identify additional properties of classifiers that can be represented by small number of groups? Recall from Section II that two rules R_1 and R_3 are transitively order-dependent on a subset of fields $\mathcal{F}$ if there exists a rule R_2 such that R_1 and R_2 are order-dependent and R_2 and R_3 are order-dependent. A maximal by inclusion subset of transitively order-dependent rules on $\mathcal{F}$ is denoted by $D(\mathcal{F})$. Note that it is possible to have several transitively order-independent sets on the same subset of fields $\mathcal{F}$. We will use a superscript to distinguish between them. A maximal subset of all rules that have no intersection in values of the fields in $\mathcal{F}$ is denoted by $I(\mathcal{F})$.

Example 10: For example, consider six rules that contain a single field: $R_1 = ([1,1])$, $R_2 = ([2,4])$, $R_3 = ([3,5])$, $R_4 = ([5,6])$, $R_5 = ([7,9])$, $R_6 = ([8,8])$. In this case we have two transitively order-dependent sets: $D_1^1 = \{R_2, R_3, R_4\}$, $D_1^2 = \{R_5, R_6\}$ and $I_1 = \{R_1\}$.

Consider k groups, where each group is based on a single field (no two groups that are based on the same field). Now perform the following operation. Assign to each group all rules. For each group $\{F_i\}$ that is based on a field F_i compute $I(\{F_i\})$ and remove this subset of rules from all groups. In this case each group will contain the same set of rules that belong to different transitively order-dependent sets. We will call this set of rules a *transitively order-dependent kernel*. Clearly, adding additional groups that are based on the same or different fields only reduces the size of this kernel. A complexity of representation of this kernel actually defines a number of groups β that are required to represent it. Thus, in a lot cases a number of groups is significantly lower.

When the sets of fields required for order-independence in the different groups are pairwise disjoint, we can reduce 2-MGR to SetCover, similarly to Theorem 5. But in this case we create $k + \binom{k}{2} = k(k+1)/2$ groups, where the second term consists of all groups based on two fields.

Theorem 7: If sets of fields required for order-independence in different groups are pairwise disjoint, 2-MGR is reducible to SetCover in $O(\frac{k(k+1)}{2} \cdot N^2)$ time and has an approximation factor of $2 \cdot \ln(N) + 1$.

In Section VI-B2 we propose a simple heuristic to resolve Problem 2 and evaluate its performance in Section VIII.

2) Representation of a Maximal Number of Rules: For the case of $k = 2$, the 2-MRC problem can be approximated with factor $O(\log \log N)$ [7]. Consider the l-MRC problem in the case of $l = k$, i.e., finding a maximal subset of rules order-independent on all k fields. In this special case, the problem reduces to the maximum independent set problem where each rule represents a node and an edge between two rules exists iff they intersect by some field.

In Theorem 5, we showed how the FSM problem reduces to SetCover; in order to find an independent set of rules based on a subset of the fields, we must make sure that each pair of rules does not intersect on at least one of the fields and defined the corresponding coverage problem. We considered the coverage of a field as the set of pairs of rules it can separate. Unfortunately, it is not straightforward to make use of the same idea in order to reduce the l-MRC problem to the MSC because a field that separates the maximal number of pairs of rules does not necessarily yield the maximal size of an independent set of rules.

For example, consider the following classifier $\mathcal{K} = (R_1, R_2, R_3, R_4)$ with four rules of two fields: $R_1 = ([0,1],[0,0])$, $R_2 = ([2,3],[1,1])$, $R_3 = ([0,1],[2,2])$, $R_4 = ([2,3],[0,3])$. There are four pairs of rules that do not intersect in the first field. These are pairs (R_1, R_2), (R_1, R_4), (R_2, R_3), (R_3, R_4). Likewise, there are three pairs of rules with disjoint second field: (R_1, R_2), (R_1, R_3) and (R_2, R_3). For this classifier, in the 1-MRC problem we should select the second field to obtain a maximal independent subset of rules although it distinguishes between fewer pairs of rules than the first field. Based only on the first field, the maximal set of independent rules includes two rules ($\{R_1, R_2\}, \{R_1, R_4\}, \{R_2, R_3\}$, or $\{R_3, R_4\}$). Adding the second field, we get an independent set of three rules $\{R_1, R_2, R_3\}$ because we can separate all pairs of rules in it, so the size of the maximal independent set is not determined by the number of disjoint pairs of rules.

Nevertheless, we can use an algorithm for the l-MSC problem as a heuristic to solve the l-MRC problem. We give the pairs of rules each field can separate as input to an algorithm that solves l-MSC.

Problem 7 (Maximum Set Coverage (l-MSC)): Given a set of elements $\mathcal{U} = \{1, 2, \ldots N\}$ (the universe), a set $\mathcal{S}$ of k sets whose union equals $\mathcal{U}$, and a positive number $l \leq k$, select at most l sets of $\mathcal{S}$ such that as many elements of $\mathcal{U}$ as possible are covered (the union of selected sets has maximal size).

A variant of Algorithm 3 that stops after l subsets are added solves the l-MSC problem and has an approximation factor of $1 - \frac{1}{e} + o(1)$ [17]. To solve (β, l)-MRC, we can run l-MRC; once any unassigned rule "intersects" with any rule assigned to the current group a new group is opened if the total number of groups does not exceed β. The algorithm for l-MGR is the same as for (β, l)-MRC but now we stop to create new groups once we covered all rules of the original classifier. The proposed heuristic for the l-MRC problem and its variant for (β, l)-MRC have worst-case guaranteed lookup time $O(\log N)$ and a linear space requirement when $l = 2$. In practice there is an excess of information that preserves order-independence on a relatively small subset of bits (see Section VIII)

VII. Classifier Configuration

A. Standardization of Policy Classifiers

We group classifiers into two categories: those required for switching and those implementing service-level-agreement (SLA) policies. The former type can be changed frequently, while the latter is typically static. Dynamic updates for the latter type are significantly less important and we can consider offline computation for more efficient representations. We suggest to specify certain classifier characteristics (that can be computed offline) explicitly as part of a classifier configuration. The following traits (or variations thereof) are important: (1) maximal size of the order-independent part; (2) minimal subset of fields that preserve order-independence; (3) minimal number of order-independent groups based on at most two fields; (4) assignment of rules to a predefined number of groups based on at most two fields. With these, optimal implementation can be chosen under the constraints of a configured network element.

B. Dynamic Updates

Suppose that classifier represents packet flows that can change frequently. It is straightforward to remove a rule from the order-independent part $\mathcal{I}$; for the order-dependent part $\mathcal{D}$, it depends on the representation. Insertion is more complicated. If an inserted rule R is order-dependent with the current $\mathcal{I}$, insert R to $\mathcal{D}$. If $\mathcal{D}$ is full, recompute $\mathcal{I}$ and $\mathcal{D}$ for the new set. If $\mathcal{D}$ is still full, reject R. If R is order-independent with $\mathcal{I}$ on k fields and does not increase the subset of fields in FSM, or there is an assignment to a groups in MGR without increasing the number of order-independent groups, add R to $\mathcal{I}$. To avoid long delays, recomputation of $\mathcal{I}$ and $\mathcal{D}$ can start in the background. There is a tradeoff between current optimality of $\mathcal{I}$ representation and delay before inserting the next rule.

We can increase the chance to successfully insert a new rule in $\mathcal{I}$. Recall that for order-independent classifiers we must check all k fields for a matched rule to avoid false-positive matches on a subset of fields. Consider an additional parameter C: the maximal number of rules that can be checked for false-positives in a line rate. Suppose an inserted rule is order-independent with each rule in the current $\mathcal{I}$ on k fields and increases the number of fields for FSM or requires more groups to keep all groups in MGR order-independent. We can still assign R to $\mathcal{I}$ if at most C rules have to be checked for false positives, increasing $|\mathcal{I}|$ by up to a factor of C.

It is also desirable to support modifications in existing rules. If a rule in $\mathcal{I}$ is only modified in fields not required for order-independence, we can simply change it in additional memory. If the change is in one of the fields required for order-independence, we have to make sure this rule does not intersect other rules in $\mathcal{I}$ after the change. If so, we can simply update this rule in the representation of $\mathcal{I}$, and if it now intersects with some other rule, we can move this modified rule to $\mathcal{D}$. In an offline process, we can recompute field selection to account for the recent new rules. As above, modification of a rule in $\mathcal{D}$ depends on the representation of $\mathcal{D}$.

Example 11: Consider a classifier $\mathcal{K} = (R_1, R_2, R_3)$ with three fields of four bits each and rules $R_1 = ([1,3],[4,8],[1,5])$, $R_2 = ([7,7],[1,8],[4,5])$, $R_3 = ([4,5],[6,9],[4,6])$. The first field suffices for order-independence, so we have $\mathcal{K}^{-\{2,3\}} = (([1,3]),([7,7]),([4,5]))$ and $\mathcal{I} = \{R_1, R_2, R_3\}$. We try to insert $R_4 = ([2,4],[2,2],[3,3])$; it is order-independent with $\mathcal{I}$ but requires the second field for order-independence with R_1 and R_3. In this (unlucky) case we can still use only the first field for $\mathcal{I}$ and test R_4 additionally if either R_1 or R_3

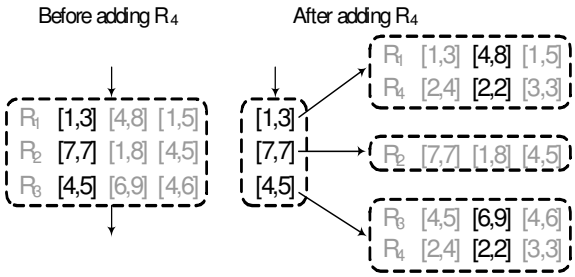


Fig. 7: Example 11. Left: a multi-group representation of $\mathcal{I} = \{R_1, R_2, R_3\}$. Right: a representation of the two-step lookup procedure for $C \geq 2$ and recently inserted R_4. Fields to be checked in a group lookup are in black; those irrelevant in this step, in grey.

is matched (this requires $C \geq 2$). Note that it is not necessary to match R_4 if R_2 is matched since it is order-independent with R_2 on the first field (see Figure 7).

VIII. Simulations

To validate our scheme, we have run simulations on 12 classifiers from Classbench [11] (generated with real parameters), each with $\approx$ 50K rules on 6 fields, and on 5 real life classifiers provided by Cisco Systems. In Figure 1 and Table I, we have already presented the space savings resulting from the FSM approach when adding new fields to these classifiers; we have shown that order-independence greatly simplifies adding new fields to classifiers. In Table II, we compared the proposed FSM approach with MinDNF heuristics, showing that FSM algorithms can have a much greater effect. The next set of simulations deals with maximal order-independent sets. We use the greedy algorithm for the l-MRC problem (see Section VI-B2) to find a maximal order-independent set on all k fields. Then we apply the FSM problem on the computed maximal order-independent set. To simulate Problem 2 (l-MGR) with one- and two-field order-independent groups, we use the algorithm proposed in Section VI-B2.

The results are summarized in Tables III and IV. In most cases, the vast majority of rules are covered by very few groups of order-independent rules. In some cases the maximal order-independent set constructed with a subset of fields $\mathcal{F} = \{0, 1\}$ is larger than such a set constructed with all fields; this is due to a few rules with large intervals in the first two fields that have been taken up to the set because they are independent on some other field and prevented more rules to join the set later. For example, a rule of the form

0.0.0.0/0 0.0.0.0/0 1234 : 1234 0 : 65535 0x00/0x00 0x0000/0x0000

would block all further rules with source port 1234 if this port has not appeared earlier but would not be taken if only the IP addresses had been taken into account. Thus, interestingly, a greedy approach is not necessarily monotonic as the number of fields taken into account increases.

The effect explained in Example 5 is also seen in simulations. In many cases we see many very small groups (of size ≤ 2 or ≤ 5) created by general rules at the bottom of the lists; in practice it makes sense to send these rules to $\mathcal{D}$. Table III also shows that a maximal order-independent subset mostly takes care of the small groups, as very few such groups are left when running l-MGR on the resulting maximal order-independent set.

The majority of the proposed approximate algorithms are greedy without backtracking (the variants of GreedySetCover algorithm) with $|\mathcal{U}| = O(N^2)$, the running time of these algorithms is $O(kN^2)$, where N is a number of rules and k is a number of fields in a classifier. As a result we do not evaluate a computational time of the proposed algorithms.

In this work, we have introduced an abstraction layer to represent a part of classifier mostly independent of specific implementations in network elements. Actual lookup time and power measurements would be strongly dependent on actual representation of the chosen fields in the abstraction layer. Table IV clearly shows that the new abstraction layer we propose can bring additional savings in space for already existing representations. In case when the classifier is stored in TCAM, Table IV also shows drastic savings in power consumption. Note that because of the transparency to a majority of specific implementations, the proposed abstraction is not an alternative but rather a way to improve specific implementations independent from them.

IX. Related Work

Research towards efficient implementations of packet classifiers falls into two main categories: algorithmic solutions (usually software based) and TCAM-based solutions. Algorithmic solutions mainly rely on one of three techniques: decision trees, hashing, or coding-based compression. The works [19], [49], [56] suggest how to partition the multi-dimensional rule space. Possible matching rules are found by tracing a path in a decision tree. Techniques to balance the partition in each node exist, but rule replication often cannot be avoided; see a related approach in [60]. There is an inherent tradeoff between space and time complexities in these approaches. Song and Turner's ABC algorithm for filter distribution offers higher throughput with lower memory overhead and can tune the implementation for better time complexity or better space complexity [51]. The works [14], [21] discuss hash-based solutions to match a packet to its possible matching rules. Efficient coding-based representations are shown in [31], [41], [43], [44].

Other works discuss efficient TCAM implementations. TCAMs do not natively support ranges, and a range often cannot be represented by a single TCAM entry. Two popular range encoding schemes are the binary encoding [53] and the SRGE encoding [6]. In the binary encoding, a W-bit range is encoded as a union of disjoint subtrees in the binary tree of 2^W leaves. Each subtree is represented by a single prefix TCAM entry. With this scheme, the maximal number of entries to encode a range (worst-case expansion) is $2(W-1)$. For example, for $W = 5$ the range $[16, 23]$ is encoded by a single entry $10***$ while the range $[1, 30]$ is described by $2W - 2 = 8$ entries 00001, 0001*, 001**, 01***, 10*** 110**, 1110*, 11110 that represent the eight subtrees of $[1], [2, 3], [4, 7], [8, 15], [16, 23][24, 27], [28, 29], [30]$, respectively. The SRGE encoding can improve the worst-case bound to $2(W-2)$ by representing values in Gray code. In

	Total rules	k-MRC size	k-MRC FSM	k-MRC $\{0,1\}$	MGR on the entire set: 1-MGR all	95%	99%	≤ 2	≤ 5	2-MGR all	95%	99%	≤ 2	≤ 5	MGR on the k-MRC result: 1-MGR all	95%	99%	≤ 2	≤ 5	2-MGR all	95%	99%	≤ 2	≤ 5
acl1	49870	49779	0, 1, 4	49768	16	1	1	6	9	12	1	1	5	7	5	1	1	1	2	2	1	1	0	0
acl2	47276	44178	0, 1, 3, 4	43819	67	5	13	19	32	39	2	5	10	20	17	4	7	1	2	6	1	2	1	1
acl3	49859	47674	0, 1, 2, 3, 4	46114	31	3	5	11	20	18	2	3	7	10	11	3	4	1	2	6	1	3	1	1
acl4	49556	46670	0, 1, 2, 3, 4	40518	42	8	14	6	15	16	3	5	3	5	21	7	10	0	3	6	2	3	0	0
acl5	40362	38962	0, 1, 3, 4	22725	43	18	27	2	6	11	4	6	1	1	37	17	25	0	2	10	4	6	1	1
fw1	47778	43675	0, 1, 2, 3, 4	44713	71	2	5	18	42	41	2	3	10	19	7	2	2	2	2	4	1	2	0	1
fw2	48885	48826	0, 1, 2, 4	48755	20	3	3	15	15	12	1	1	5	9	7	3	3	2	3	2	1	1	0	0
fw3	46038	41615	0, 1, 2, 3, 4	40581	101	2	17	42	57	56	2	7	11	28	17	2	2	2	4	7	1	2	0	0
fw4	45340	42857	0, 1, 2, 3, 4	43912	109	2	40	9	28	57	1	17	10	16	20	2	2	2	3	8	1	1	0	1
fw5	45723	39794	0, 1, 2, 3, 4	39007	94	2	13	38	58	49	2	8	6	21	8	2	2	2	3	5	1	2	1	2
ipc1	49840	48294	0, 1, 3, 4	48385	22	2	2	9	16	16	1	3	6	11	6	1	2	1	2	4	1	1	0	0
ipc2	50000	50000	0, 1	50000	2	2	2	0	0	1	1	1	0	0	2	2	2	0	0	1	1	1	0	0
cisco1	584	538	0, 1, 3, 4	406	15	8	13	2	8	10	4	7	3	4	9	5	7	2	3	5	2	4	1	2
cisco2	269	249	0, 1, 4	246	4	2	3	1	1	2	2	2	0	0	4	2	3	1	2	2	1	2	0	1
cisco3	95	92	0, 1, 3, 4	89	5	3	5	2	3	3	2	3	1	2	4	2	4	2	2	2	1	2	0	1
cisco4	364	329	0, 1, 3, 4	324	7	3	5	2	4	4	2	3	1	1	5	2	3	2	3	3	1	2	1	2
cisco5	148	120	0, 1	120	3	2	3	0	1	2	2	2	0	0	2	2	2	0	0	1	1	1	0	0

TABLE III: Simulation results. Columns, left to right: total no. of rules $|\mathcal{K}|$; $|\mathcal{I}|$, where $\mathcal{I} = \text{MRC}(\mathcal{K}, \{0, \ldots, 5\})$; minimal size subset of fields w.r.t. which $\mathcal{I}$ is order-independent; $|\mathcal{I}'|$, where $\mathcal{I}' = \text{MRC}(\mathcal{K}, \{0, 1\})$ (i.e., order-independent w.r.t source IP and destination IP); for the one-field result $\{\mathcal{G}\} = MGR(\mathcal{K})$: total number of order-independent groups $|\{\mathcal{G}\}|$, no. of groups covering 95% and 99% of the rules, no. of groups of size ≤ 2 and of size ≤ 5; similar statistics for the two-field MGR; similar statistics for one-field and two-field MGR run on $\mathcal{I}$.

	Original: Binary Kbits	Blocks	Wt	Original: SRGE Kbits	Blocks	Wt	k-MRC: Binary Kbits	Blocks	Wt	k-MRC: SRGE Kbits	Blocks	Wt	MRC, $\{0,1\}$: Binary Kbits	Blocks	Wt	MRC, $\{0,1\}$: SRGE Kbits	Blocks	Wt	MGR, 95%: Binary Kbits	Blocks	Wt	MGR, 95%: SRGE Kbits	Blocks	Wt	MGR, 99%: Binary Kbits	Blocks	Wt	MGR, 99%: SRGE Kbits	Blocks	Wt
acl1	7922.1	68	20.1	7656.0	66	19.5	2647.8	34	10.1	2559.1	33	9.8	2121.3	34	10.1	2050.4	33	9.8	2122.5	35	10.4	2051.6	34	10.1	2122.5	35	10.4	2051.6	34	10.1
acl2	11289.1	97	28.7	11289.1	97	28.7	5615.9	51	15.1	5615.9	51	15.1	3587.5	52	15.4	3587.5	52	15.4	3124.2	49	14.5	3124.2	49	14.5	3034.3	49	14.5	3034.3	49	14.5
acl3	10771.6	92	27.2	10571.2	91	26.9	6775.5	50	14.8	6651.1	49	14.5	3537.6	50	14.8	3464.1	49	14.5	2931.9	47	13.9	2880.7	46	13.6	2880.2	47	13.9	2827.1	46	13.6
acl4	10079.3	87	25.8	9904.9	85	25.2	6430.7	48	14.2	6322.5	47	13.9	4215.7	52	15.4	4130.9	51	15.1	2794.1	44	13.0	2751.3	43	12.7	2712.8	44	13.0	2666.9	43	12.7
acl5	6121.9	53	15.7	6121.9	53	15.7	3152.2	29	8.6	3152.2	29	8.6	3797.4	39	11.5	3797.4	39	11.5	1747.1	27	8.0	1747.1	27	8.0	1651.5	27	8.0	1651.5	27	8.0
fw1	19454.1	167	49.4	19438.6	166	49.1	11976.3	87	25.8	11961.1	87	25.8	5727.9	87	25.8	5712.4	87	25.8	5213.5	84	24.9	5213.5	84	24.9	5203.0	84	24.9	5202.7	84	24.9
fw2	10866.8	93	27.5	10866.8	93	27.5	5076.4	47	13.9	5076.4	47	13.9	2922.7	47	13.9	2922.7	47	13.9	2906.9	48	14.2	2906.9	48	14.2	2906.9	48	14.2	2906.9	48	14.2
fw3	15090.0	129	38.2	15073.1	129	38.2	9348.0	68	20.1	9332.0	68	20.1	4661.8	69	20.4	4645.0	68	20.1	4110.7	66	19.5	4110.7	66	19.5	4034.9	65	19.2	4034.2	65	19.2
fw4	33500.3	286	84.7	33368.8	285	84.4	20427.4	147	43.5	20341.2	146	43.2	9491.1	147	43.5	9441.1	146	43.2	8899.5	142	42.0	8869.9	142	42.0	8853.8	143	42.3	8821.4	143	42.3
fw5	12478.5	107	31.7	12445.4	107	31.7	7810.9	57	16.9	7785.5	57	16.9	4007.0	58	17.2	3985.8	57	16.9	3369.6	54	16.0	3365.3	54	16.0	3334.9	54	16.0	3329.7	54	16.0
ipc1	8041.1	69	20.4	7924.2	68	20.1	3871.2	36	10.7	3815.8	35	10.4	2283.1	36	10.7	2250.2	35	10.4	2157.3	36	10.7	2126.1	35	10.4	2157.3	36	10.7	2126.1	35	10.4
ipc2	5859.4	51	15.1	5859.4	51	15.1	1562.5	26	7.7	1562.5	26	7.7	1562.5	26	7.7	1562.5	26	7.7	1562.5	27	8.0	1562.5	27	8.0	1562.5	27	8.0	1562.5	27	8.0
cisco1	78.3	1	0.3	78.3	1	0.3	40.6	1	0.3	40.6	1	0.3	40.0	1	0.3	40.0	1	0.3	22.3	1	0.3	22.3	1	0.3	20.4	1	0.3	20.4	1	0.3
cisco2	68.6	1	0.3	68.6	1	0.3	24.4	1	0.3	24.4	1	0.3	20.7	1	0.3	20.7	1	0.3	19.0	1	0.3	19.0	1	0.3	18.4	1	0.3	18.4	1	0.3
cisco3	11.1	1	0.3	11.1	1	0.3	5.4	1	0.3	5.4	1	0.3	3.5	1	0.3	3.5	1	0.3	3.1	1	0.3	3.1	1	0.3	2.9	1	0.3	2.9	1	0.3
cisco4	79.7	1	0.3	79.7	1	0.3	40.4	1	0.3	40.4	1	0.3	26.1	1	0.3	26.1	1	0.3	21.5	1	0.3	21.5	1	0.3	21.0	1	0.3	21.0	1	0.3
cisco5	19.6	1	0.3	19.6	1	0.3	7.6	1	0.3	7.6	1	0.3	7.6	1	0.3	7.6	1	0.3	5.5	1	0.3	5.5	1	0.3	5.2	1	0.3	5.2	1	0.3

TABLE IV: Simulation results: sizes in Kbits corresponding to results shown in Table III. TCAM space and power consumption figures are based on 80-bit TCAM cells with blocks of size 2K cells [35] and power consumption of 1.85 Wt per Mbit of the final TCAM (in full blocks) [2].

this encoding, TCAM entries are not necessarily prefixes and do not necessarily represent disjoint subsets of the range. For example, for $W = 5$ the range $[1, 30]$ can be encoded by *0001, *001*, *01**, *1***. Both schemes encode a multi-field range with a number of entries that is exponential in the number of fields k with upper bounds of $(W-2)^k$ and $(W-4)^k$, respectively. When entries that negatively encode the complement of the ranges can be used (deny entries), the upper bound is improved to W entries [45], [46]. The works [45], [47] suggest schemes to encode a k-field range with complexity linear in k. However, they apply only to encoding a single rule; to encode a classifier with more rules, one has to change the conventional TCAM architecture.

Several works have tried to reduce the number of TCAM entries for a given classifier by relying on heuristics [8], [36]. These techniques include encoding the most common ranges with additional bits [10], [34], applying block permutations [59], designing tree-based architectures [55], and redundancy removal [33]. While many such heuristics could be improved with randomization [32], they usually limit the number of fields in a rule or perform badly as the number of fields increases. Schemes for representing rule updates by timestamp-based TCAM range rules have been shown in [48]. Efficient schemes for classification and update supporting that rely on rule disjointness have been proposed in [5], [52], [58].

Geometrically speaking, a rule with multiple prefixes represents a multidimensional hyper-rectangle in the search space, a classifier is a set of hyper-rectangles with assigned priorities, and the classification problem is to find the highest priority hyper-rectangle covering a given point. Point location problem involves finding the enclosing region of a point, given the set of non-overlapping hyper-rectangles. Complexity bounds for the point location problem 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 [39]. In the general case hyper-rectangles corresponding to rules of a classifier can overlap making packet classification at least as hard as a point location. In this paper we exploit order-independence to reduce the number of classification fields participating in classification. For instance, reducing a number of fields in representations of classifiers to two corresponds to point location problem in the two-dimensional space that leads to representations with a logarithmic lookup time and (near-)linear space [9], [16], [25]. Exploiting order-

independence to improve power and lookup time is discussed in [3], [4]. However, we know of no prior work that considers order independence specifically to reduce classification width and support additional expressiveness. Whereas power optimization is out of scope of this paper, in the TCAM case, assuming the power consumption linearly depends on the number of TCAM cells (participated in lookup), reducing the classification width implicitly results in an improved power consumption (see Table IV). Multi-group representations to reduce power consumption have appeared at [35], [50], while the space-time tradeoff for representation of hierarchical classifiers has been considered in [15], [23], [27]. Minimization of memory requirements for multiple instances of a given classifier has been explored in [29]. The relation between packet classifiers and optimization of Boolean expressions was recently studied in [28]. We also note that exploiting order-independence can significantly simplify splitting of a classifier over several network elements [20], [22].

In this paper we introduce an abstraction layer that exploits order-independence to represent a part of classifier mostly independent of specific implementations in network elements. This layer only defines a subset of fields that should be involved in the lookup process to order-independent parts of classifiers and as a result is applicable to both SW- and TCAM-based solutions; the remaining rules can be represented in the traditional ways [6], [10], [33] (to list just a few). In particular, if we remove the redundant rules [33] in an order-independent classifier, it remains order-independent. We can also use bit vectors to represent the common rules [10] after we reduce the width of a classifier or use Gray encoding for ranges [6] representing the chosen fields of a classifier.

X. Conclusion

Packet classification depends on the number of entries N in a classifier and the width W of its rules. In fact, for a given number of entries width is the primary factor for lookup complexity. Once we pass to *multi*-field classification, width, and hence complexity, tend to grow significantly. This is especially the case for expressive classification with ranges. In practice, rules contain excessive information to distinguish between N entries. Various techniques have been proposed to mine similar patterns across table entries and represent and evaluate them efficiently (see, e.g., [41]).

In this work, we take a different route, identifying new properties of classifiers that let us ignore superfluous information in classification lookup. Our proposed concept of order-independence simplifies classifier matching by splitting the problem into two subproblems: (a) the rule selection problem which is based only on parts of rules (not their entire width) and (b) the matching of the selected rule which is based only on the remainder of the fields not considered in (a). As we have demonstrated, the proposed classifier representations can significantly outperform dedicated space minimization approaches. While the proposed techniques are motivated by packet classification problems and demonstrated thereon, we believe that our simple yet effective concepts can be reused for significant improvements also in neighboring areas (e.g., matching in databases or data mining) and can also open new lines of research in other areas (e.g., Boolean minimization).

References

[1] E. Allender, L. Hellerstein, P. McCabe, T. Pitassi, and M. E. Saks. Minimizing DNF formulas and AC^0_d circuits given a truth table. In *IEEE Conference on Computational Complexity*, 2006.

[2] B. Agrawal and T. Sherwood. Modeling TCAM Power for Next Generation Network Devices. In *IEEE International Symposium on Performance Analysis of Systems and Software*, 2006.

[3] T. Banerjee-Mishra, S. Sahni, and G. S. Seetharaman. PC-DUOS+: A TCAM Architecture for Packet Classifiers. *IEEE Trans. Computers*, 63(6):1527–1540, 2014.

[4] T. Banerjee-Mishra, S. Sahni, and G. S. Seetharaman. PC-TRIO: A power efficient TCAM architecture for packet classifiers. *IEEE Trans. Computers*, to appear.

[5] A. Bremler-Barr, D. Hay, and D. Hendler. Layered interval codes for TCAM-based classification. In *IEEE Infocom*, 2009.

[6] A. Bremler-Barr and D. Hendler. Space-efficient TCAM-based classification using Gray coding. *IEEE Trans. Computers*, 61(1):18–30, 2012.

[7] P. Chalermsook and J. Chuzhoy. Maximum independent set of rectangles. In *ACM-SIAM SODA*, 2009.

[8] Y.K. Chang, C.I. Lee, and C.C. Su. Multi-field range encoding for packet classification in TCAM. In *IEEE Infocom Mini-Conference*, 2011.

[9] R. Cole. Searching and Storing Similar Lists. *J. Algorithms*, 7(2):202–220, 1986.

[10] H. Che, Z. Wang, K. Zheng, and B. Liu. DRES: Dynamic range encoding scheme for TCAM coprocessors. *IEEE Trans. Computers*, 57(7):902–915, 2008.

[11] ClassBench: A packet classification benchmark. http://www.arl.wustl.edu/classbench/.

[12] Configuring IP ACLs. http://www.cisco.com/en/US/docs/switches/datacenter/sw/4_1/nx-os/security/configuration/guide/sec_ipacls.pdf.

[13] Configuring TCAM Carving. http://www.cisco.com/en/US/td/docs/switches/datacenter/nexus6000/sw/security/6x/b_6k_Security_Config_6x/b_6k_Security_Config_6x_chapter_01110.pdf.

[14] S. Dharmapurikar, H. Song, J. S. Turner, and J. W. Lockwood. Fast packet classification using bloom filters. In *ACM/IEEE ANCS*, 2006.

[15] A. Dixit, K. Kogan, P. Eugster. Composing Heterogeneous SDN Controllers with Flowbricks. In *IEEE ICNP*, 2014.

[16] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal Point Location in a Monotone Subdivision. *SIAM J. Comput.*, 15(2):317–340, 1986.

[17] U. Feige. A threshold of ln *n* for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[18] A. Feldman and S. Muthukrishnan. Tradeoffs for packet classification. In *IEEE Infocom*, 2000.

[19] P. Gupta and N. McKeown. Classifying packets with hierarchical intelligent cuttings. *IEEE Micro*, 20(1):34–41, 2000.

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

[21] Y. Kanizo, D. Hay, and I. Keslassy. Optimal fast hashing. In *IEEE Infocom*, 2009.

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

[23] A. Kesselman, K. Kogan, S. Nemzer, and M. Segal. Space and speed tradeoffs in TCAM hierarchical packet classification. *J. Comput. Syst. Sci.*, 79(1):111–121, 2013.

[24] S. Khot and R. Saket. Hardness of minimizing and learning DNF expressions. In *IEEE FOCS*, 2008.

[25] D. G. Kirkpatrick. Optimal Search in Planar Subdivisions. *SIAM J. Comput.*, 12(1):28–35,1983.

[26] J. M. Kleinberg and É. Tardos. *Algorithm design*. Addison-Wesley, 2006.

[27] K. Kogan, A. Dixit, P. Eugster. Serial Composition of Heterogeneous Control Planes. In *ONS*, 2014.

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

[29] K. Kogan, S. I. Nikolenko, P. Eugster, E. Ruan. Strategies for Mitigating TCAM Space Bottlenecks. In *IEEE Hot Interconnects*, 2014.

[30] K. Kogan, S. I. Nikolenko, O. Rottenstreich, W. Culhane, P. Eugster. SAX-PAC (Scalable And eXpressive PAcket Classification). In *ACM SIGCOMM*, 2014.

[31] A. Korösi, J. Tapolcai, B. Mihalka, G. Mészáros, G. Rétvári. Compressing IP Forwarding Tables: Realizing Information-Theoretical Space Bounds and Fast Lookups Simultaneously. In *IEEE ICNP*, 2014.

[32] N. Lesh and M. Mitzenmacher. Bubblesearch: A simple heuristic for improving priority-based greedy algorithms. *Inf. Process. Lett.*, 97(4):161–169, 2006.

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

[34] H. Liu. Efficient mapping of range classifier into Ternary-CAM. In *IEEE Hot Interconnects*, 2002.

[35] Y. Ma and S. Banerjee. A smart pre-classifier to reduce power consumption of tcams for multi-dimensional packet classification. In *ACM SIGCOMM*, 2012.

[36] C. R. Meiners, A. X. Liu, and E. Torng. Bit weaving: A non-prefix approach to compressing packet classifiers in TCAMs. *IEEE/ACM Trans. Netw.*, 20(2):488–500, 2012.

[37] Netlogic Microsystems. Content addressable memory. http://www.netlogicmicro.com.

[38] OpenFlow 1.3 specification, 2012. http://www.openflow.org/wk/index.php/OpenFlow_1_3_proposal.

[39] M. H. Overmars and A. Frank van der Stappen. Range searching and point location among fat objects. *J. Algorithms*, 21(3):629–656, 1996.

[40] P. Gupta and N. McKeown. Packet Classification on Multiple Fields. In *ACM SIGCOMM*, 1999.

[41] G. Rétvári, J. Tapolcai, A. Korösi, A. Majdán, and Z. Heszberger. Compressing ip forwarding tables: towards entropy bounds and beyond. In *ACM SIGCOMM*, 2013.

[42] J.A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12(1):23–41, 1965.

[43] O. Rottenstreich, A. Berman, Y. Cassuto and I. Keslassy. Compression for fixed-width memories. In *IEEE ISIT*, 2013.

[44] O. Rottenstreich, M. Radan, Y. Cassuto, I. Keslassy, C. Arad, T. Mizrahi, Y. Revah, and A. Hassidim. Compressing forwarding tables for datacenter scalability. *IEEE Journal on Selected Areas in Communications*, 32(1):138–151, 2014.

[45] O. Rottenstreich and I. Keslassy. Worst-Case TCAM Rule Expansion. In *IEEE Infocom Mini-Conference*, 2010.

[46] O. Rottenstreich and I. Keslassy. On the Code Length of TCAM Coding Schemes. In *IEEE ISIT*, 2010.

[47] O. Rottenstreich, I. Keslassy, A. Hassidim, H. Kaplan, and E. Porat. On finding an optimal TCAM encoding scheme for packet classification. In *IEEE Infocom*, 2013.

[48] T. Mizrahi, O. Rottenstreich and Y. Moses. TimeFlip: Scheduling Network Updates with Timestamp-based TCAM Ranges. In *IEEE Infocom*, 2015.

[49] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet classification using multidimensional cutting. In *ACM SIGCOMM*, 2003.

[50] M. Somasundaram. Memory and power efficient mechanism for fast table lookup.

[51] H. Song and J. S. Turner. ABC: Adaptive binary cuttings for multidimensional packet classification. *IEEE/ACM Trans. Netw.*, 21(1):98–109, 2013.

[52] V. Srinivasan, S. Suri, and G. Varghese. Packet classification using tuple space search. In *ACM SIGCOMM*, 1999.

[53] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. In *ACM SIGCOMM*, 1998.

[54] C. Umans. The minimum equivalent DNF problem and shortest implicants. *J. Comput. Syst. Sci.*, 63(4):597–611, 2001.

[55] B. Vamanan and T. N. Vijaykumar. TreeCAM: decoupling updates and lookups in packet classification. In *ACM CoNEXT*, 2011.

[56] B. Vamanan, G. Voskuilen, and T. N. Vijaykumar. EffiCuts: optimizing packet classification for memory and throughput. In *ACM SIGCOMM*, 2010.

[57] G. Varghese. *Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices*. Morgan Kaufmann, 2005.

[58] Z. Wang, H. Che, M. Kumar, and S. K. Das. CoPTUA: Consistent policy table update algorithm for TCAM without locking. *IEEE Trans. Computers*, 53(12):1602–1614, 2004.

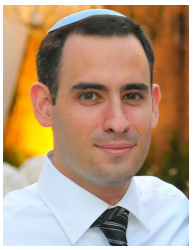
[59] R. Wei, Y. Xu, and H. J. Chao. Block permutations in Boolean space to minimize TCAM for packet classification. In *IEEE Infocom Mini-Conference*, 2012.

[60] X. Zhao, Y. Liu, L. Wang, and B. Zhang. On the aggregatability of router forwarding tables. In *IEEE INFOCOM*, 2010.

[61] Cisco 12000 Series Internet Router Architecture: Line Card Design. http://www.cisco.com/c/en/us/support/docs/routers/12000-series-routers/47242-arch12000-lcdesign.html.

Kirill Kogan Kirill Kogan is a Research Assistant Professor at IMDEA Networks Institute. He received his PhD from Ben-Gurion University (Israel) at 2012. He is a former Technical Leader at Cisco Systems, where he worked during 2000-2012. He was a Postdoctoral Fellow at University of Waterloo and Purdue University during 2012-2014. His current research interests are in design, analysis, and implementation of networked systems, broadly defined (in particular network processors, switch fabrics, packet classification, network management, service architecture, cloud computing).

Sergey Nikolenko Sergey Nikolenko is a Senior Researcher at the Laboratory for Internet Studies, National Research University Higher School of Economics, and Laboratory of Mathematical Logic at the Steklov Institute of Mathematics at St. Petersburg. He received M.Sc. summa cum laude from St. Petersburg State University at 2005 and Ph.D. from the Steklov Institute of Mathematics at St. Petersburg at 2009. His research interests include networking algorithms and systems, machine learning and probabilistic inference, bioinformatics, and theoretical computer science, with research and educational projects funded by major companies and research funds such as CRDF, INTAS, Mail.Ru, RFBR, RAS, and Russian governmental programs.



Ori Rottenstreich Ori Rottenstreich is a Postdoctoral Research Associate at the Department of Computer Science, Princeton University, Princeton, NJ. He received B.S. in computer engineering (summa cum laude) in 2008 and Ph.D. degree in 2014 from the Electrical Engineering Department of the Technion, Haifa, Israel. He is a recipient of the Rothschild Yad-Hanadiv postdoctoral fellowship, the Google Europe Fellowship in Computer Networking, the Andrew Viterbi graduate fellowship, the Jacobs-Qualcomm fellowship, the Intel graduate fellowship and the Gutwirth Memorial fellowship. He also received the Best Paper Runner Up Award at the IEEE Infocom 2013 conference.

William Culhane William Culhane is a computer science graduate student at Purdue University. His interests include distributed systems, especially with regard to data management and processing algorithms. His recent work focuses on optimal aggregation overlays based on mathematical modelling. He has a bachelor's degree from the Ohio State University and a master's degree from Purdue University.

Patrick Eugster Patrick Eugster is an Associate Professor of Computer Science at Purdue University and a Visiting Professor at Technical University of Darmstadt. He is interested in networked distributed systems as well as programming languages and models, and in particular in the intersection of these topics. Patrick holds M.S. and Ph.D. degrees from EPFL. His research has been recognized by several awards, including CAREER (2007) from the US National Science Foundation (NSF) and Consolidator (2013) from the European Research Council (ERC). Patrick is a member of the Computer Science Study Panel of the Defense Advanced Research Programs Agency (DARPA) and the Purdue University Innovator Hall of Fame. His research has been funded by several companies including Google, Cisco, NetApp, and Northrop Grumman, as well as public funding agencies including NSF, DARPA, ERC, DFG (German Research Foundation), and SNF (Swiss National Research Foundation).