Optimal Rule Caching and Lossy Compression for Longest Prefix Matching

Ori Rottenstreich and János Tapolcai

Abstract—**Packet classification is a building block in many network services such as routing, filtering, monitoring and policy enforcement. In commodity switches, classification is often performed by memory components of various rule matching patterns (longest prefix match, ternary matches, exact match etc.). They are fast but expensive and power-hungry memory components with power consumption proportional to the memory size. In this paper we study the applicability of *lossy compression* to create packet classifiers requiring much less memory than the theoretical size limits of the semantically-equivalent representations, enabling significant reduction in their cost and power consumption. This study focuses on longest prefix matching. Our objective is to find a limited-size longest prefix match classifier that can correctly classify a high portion of the traffic so that it can be implemented in commodity switches with classification modules of restricted size. Concentrating on prefix classifiers, we develop optimal dynamic-programming algorithms for several versions of the problem and describe how to treat the small amount of traffic that cannot be classified. We generalize our solutions for a wide range of classifiers with different similarity metrics. We evaluate their performance on real classifiers and traffic traces and show that in some cases we can reduce a classifier size by orders of magnitude while still classifying almost all the traffic correctly.**

I. Introduction

Packet classification is a core function behind many network services such as forwarding, routing, filtering, intrusion detection, accounting, monitoring, load-balancing and policy enforcement. In recent years there has been a rapid growth in the size and power consumption of classifiers and routing tables. This phenomenon, driven by the increasing number of hosts and the appearance of more detailed network policies, results in a severe scalability problem [2], [3].

In commodity switches, classification often relies on TCAMs (Ternary Content Addressable Memories). TCAMs are used to perform very-high-speed, hardware accelerated table lookups for implementing rule matching but are expensive, often of a limited size, and power hungry. Their price and power consumption is known to be roughly proportional to their entries number [4]–[6]. While TCAMs support general wildcard matches, common policies considering a single header field, e.g., forwarding based on a destination address or load balancing based on source address, are often encoded in TCAMs using prefix rules [7], [8]. Likewise, classification plays an important role in recently suggested programmable switch architectures such as RMT and Intel's FlexPipe [6], [9] combining multiple match-action tables of different kinds where prefix rule tables are an inherent component.

Huffman coding [10] and the Lempel-Ziv-Welch algorithm [11] are well known lossless compression schemes, but are less suited for classifiers requiring fast classification without complicated decoding. Lossless compression of packet classifiers has been deeply investigated in the last decades [12]–[15]. The ORTC algorithm [16] achieves an optimal representation with a minimal number of prefix rules. For example, the uncompressed IPv4 routing table in software routers are stored in a few tens of MBs memory, which can typically be compressed to a few MBs with ORTC. Recent lossless compressions of classifiers can almost reach the information theoretical bounds, typically of a few hundred KBs, with the price of slightly more complicated lookup and update mechanisms [17]. To achieve a significant save in power consumption and price of the classifier in commodity switches we may need to go way beyond the information theoretical bounds of compression. For instance, representing a routing table in a few KBs can potentially make a big difference.

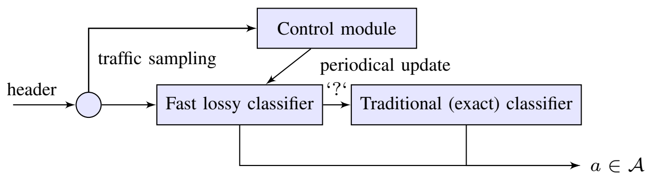
Lossy compression is a methodology for achieving higher compression ratios at the cost of loosing some information about the represented object. Lossy representations can be smaller than the Information theory based lower bounds for a lossless compression. Implementations of lossy compression schemes can be found in popular standards and applications such as JPEG (for images), MPEG (for videos) and MP3 (for audio) [18]–[20].

Our approach is motivated by an inherent property of Internet traffic. Measurements show that such traffic tends to be very biased, often following the Zipf distribution [21] and that a large portion of the traffic comes from a small number of flows [22], [23]. Accordingly, traffic matches the classification rules in a biased distribution such that some of the classifier information is very seldom useful. While it often requires knowledge on the traffic distribution, this observation has motivated adapting rule caching schemes for classifiers [24]–[28], similar to those used in CPU caching. In particular, this locality behavior was recently demonstrated in FIB (Forwarding Information Base) for IPlookup [24], [25] indicating that excellent hit rates are achieved with cache sizes order of magnitude smaller that of a complete FIB. Similar results have been demonstrated in a detailed measurement study [28] performed in a large ISP network that showed also that the size of active rules is often stable over time. While examining several cache replacement

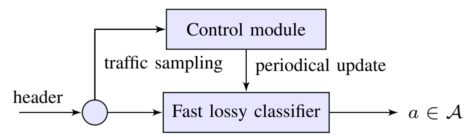
This manuscript is an extended version of [1], which appeared at the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS) '15, Oakland, CA, May 2015.

O. Rottenstreich is with the Department of Computer Science, Princeton University, NJ, USA (e-mail: orir@cs.princeton.edu).

J. Tapolcai is with the MTA-BME Future Internet Research Group, Budapest University of Technology and Economics (BME), Hungary (e-mail: tapolcai@tmit.bme.hu). J. T. is partially supported by Ericsson Hungary and the Hungarian Scientific Research Fund (grant No. OTKA 108947).



(a) Cached Classifier. It always provides a correct classification.



(b) Approximate Classifier.

Fig. 1: System architecture of the new classifiers based on *Lossy Compression*. Efficiency is achieved using fast lossy classifier with a significantly reduced number of rules.

policies, the study pointed out on a high similarity between the active sets of different routers in the same network.

This work is the first to study applying lossy compression for packet classifiers. We can see rule caching as a variant of our more general approach of lossy compression. In general a *cached classifier* has two modules: a *fast (lossy) classifier* with a small memory and a traditional *exact classifier*, see also Fig. 1(a) for the block diagram of the system architecture. The fast classifier often corresponds to a wire speed hardware device while the exact classifier to a slower component that might be software-based. A hardware implementing the fast classifier can be 5 to 100 times power efficient and faster than the exact classifier. A packet first reaches the fast classifier and is either classified correctly, or the indication '?' is obtained and the packet is sent also to the exact classifier.

Determining the content of the fast classifier is not a simple task. Due to dependencies between rules with different priorities, simply caching the most accessible rules can lead to incorrect classification [29]. Accordingly, previous caching approaches often relied on heuristic approaches without provable guarantees on their optimality. Moreover, in some systems, maintaining the classification line rate is a must and there is no option to access a complete slow memory even for a small portion of the traffic. Then, taking advantage of the fast classifier that can be smaller than any exact complete representation of the classifier, requires a novel approach. Note that some applications are less sensitive and might allow wrong classification of a small portion of the traffic, e.g., in the context of server load balancing [8]. In this case the exact classifier module is not needed, and we can rely only of the fast classifier although it sometime provides incorrect classifications. This is illustrated on Fig. 1(b). We further discuss the applicability of the schemes later in this paper.

In the paper, we focus on efficient selection of the content of the fast classifier, often much smaller than any exact representation of the classifier. We refer to that as *lossy compression of the packet classifier*. As a first step, we study one dimensional packet classifiers with only *prefix rules*. We would like to implement within a limited-size module a classifier that is similar under different metrics to the required one while satisfying various system constraints. To the best of our knowledge, this is the first time that lossy compression techniques have been studied in the context of packet classifiers. Of course going below the bounds of exact representations eliminates the option to classify all traffic correctly. We describe multiple ways to express this loss by suggesting various similarity metrics while considering various system constraints to allow and tradeoff between possible misclassifications. In the main scheme of our approach a unique action must be returned for all packets that cannot be classified due to the lossy representation of the classifier. For the unclassified packets we can then calculate the classification in an alternative traditional module. Even earlier, we provide a toy scheme that serves as a baseline for better understanding of the main scheme. In this toy scheme false classifications are allowed and can occur for a portion of the traffic. We consider additional system constraints leading to new compression optimization problems. We present algorithms that optimally solve the presented problems with the different system constraints and find the best lossy compression for any allowed number of prefix rules.

To illustrate the intuition behind lossy compression, we consider a classifier with five rules presented in Table I(a). An equivalent representation with the minimal number of prefix rules, four in this case, is the ORTC representation. It appears in Table I(b) and maps the corresponding action to every header with a width of $W = 3$ bits. Assume that the $2^W = 8$ headers appear according to a uniform distribution U and that only $n = 3$ rules are allowed in a limited-size memory.

The first (toy) scheme, called *Approximate Classification* (illustrated in Table I(c)), uses only three rules to classify correctly 7 of the 8 possible headers. With this encoding, only the header 101, which appears with probability 0.125, is mapped to an incorrect action of 1 instead of to the action 4. We say that this scheme achieves a correctness ratio of $7/8 = 0.875$. In the second (main) scheme, *Cached Classification* (in Table I(d)), all headers that cannot be classified correctly are indicated by the special action '?'. With the same number of rules (three), this scheme correctly classifies six of the eight headers, obtaining an a correctness ratio of $6/8 = 0.75$, and gives a special indication for the two headers (100 and 101) that it does not classify correctly.

The ability to deal with the incompleteness of the lossy compression can be crucial. The choice of Approximate Classification vs. Cached Classification and the method for handling incorrect classification and cache misses depends on the application. For instance, with classification errors, loops can occur in a routing application, and an application designed to filter illegal traffic might erroneously allow unwanted packets. Hence, Cached Classification would be more suitable for these applications. In some applications, e.g. load balancing among servers in a data center network, incorrect classification for a small fraction of the traffic might be tolerable. In the Cached Classification scheme, upon receiving '?' we have several choices how to obtain a correct classification. We can

prefix/length	action
100/3	3
101/3	4
01/2	2
00/2	1
11/2	1

(a) LPM Classifier with 5 rules

prefix/length	action
100/3	3
101/3	4
01/2	2
-/0	1

(b) smallest exact representation by ORTC [16] with 4 rules

prefix/length	action
100/3	3
01/2	2
-/0	1

(c) Approximate Classification, correctness ratio of 0.875 for $n = 3$ rules

prefix/length	action
10/2	'?'
01/2	2
-/0	1

(d) Cached Classification, correctness ratio of 0.75 for $n = 3$ rules

TABLE I: Illustration of the suggested encodings for limited size classifiers with a width of $W = 3$ bits and $2^W = 8$ uniformly-distributed headers. (a) shows an LPM-based classifier with 5 rules and (b) presents its ORTC compressed representation [16] with 4 rules. (c) describes an *Approximate Classification* encoding of the classifier given $n = 3$ rules. All headers besides 101 are classified correctly, yielding a correctness ratio of $7/8 = 0.875$. (d) illustrates the *Cached Classification* encoding given $n = 3$ rules. The headers 100, 101 are mapped to the unique action '?' (unclassified) while all other headers are classified correctly, yielding a correctness ratio of $6/8 = 0.75$.

calculate the classification in a slower path, i.e. by accessing a second-level larger memory or, in software-defined networks, by sending one packet header of a flow that cannot be classified to the network controller.

The suggested solutions do not directly rely on the TCAM architecture. We consider encodings of classifiers composed of prefix rules which are common especially in the context of longest prefix match (LPM) classifiers where in the case of several matching rules, a priority is given to the most specific one. Indeed, our approach can be useful to deal with limited number of allowed rules in any additional prefix-based rule memory components. As mentioned, due to their importance, several recently-suggested switch architectures include such prefix components like the BST (Binary Search Tree) memory of Intel's FlexPipe architecture with up to 64K prefix rules [9] or in corresponding tables in the RMT architecture [6].

While the focus of the paper is on longest prefix match with a single field, we demonstrate that the problem is NP-hard even for a single field with wildcard matching. Even without wildcard matching, it is also difficult to generalize the approach to an arbitrary number of fields. Consider for instance a classification based on one longest prefix match field together with an additional exact match field. In that case, the partial solutions for the tree based structure should be calculated independently for each value of the exact match field. The number of allowed rules can be divided into each of the various solutions for the different values arbitrarily. Accordingly, the complexity of a solution would increase exponentially with the number of possible values of the exact match field, eliminating the opportunity to apply the approach of this paper for such classifiers in practice.

Paper outline: In Section II we explain the terminology of the paper. Next, in Section III we define the two schemes of Approximate Classification and Cached Classification and point on their important properties in Section IV. Then in Section V and Section VI, respectively, we present optimal algorithms for the two problems. Generalizations of the approach are described in Section VII for classifiers with numerical classification values and for two-dimensional classifiers. We also show that finding optimal encodings with general rules that are not necessarily prefix is a NP-hard problem and later we study rule updates. Experimental results that demonstrate the potential of the lossy compression approach are given in Section VIII. Related work can be found in Section IX. In Section X we elaborate on the applicability of the approach to a wide range of classifiers. Proofs of the main results appear in the Appendix.

II. Model and Notation

We first formally define the terminology of this paper.

Definition 1: A **packet header** $x = (x_1, \cdots, x_W) \in \{0,1\}^W$ is defined as a W-bit string that serves as an input to the classification process.

In the main part of the paper we assume a simple case in which the classification is performed on a single field, e.g. the source or the destination IP address. Note that the typical values for W are 32 for IPv4 addresses, and 128 for IPv6. We later discuss a more general case.

Definition 2: A **prefix rule**, denoted by $S \to a$, is defined as a string $S = s_1 \ldots s_k \in \{0,1\}^k$ of length $k \leq W$ associated with an action a among a set of actions $\mathcal{A}$. A packet header $x = x_1 \ldots x_W$ is said to **match** a rule S, if and only if for all $i \in [1,k]$, $s_i = x_i$.

Definition 3: A **classification function** defines the mapping of each header to an **action** $a \in \mathcal{A}$.

Definition 4: A **prefix classifier** $C^\phi = (S^1 \to a^1, ..., S^n \to a^n)$ is an ordered set of prefix rules. It encodes a classification function ϕ such that for any packet header $x \in \{0,1\}^W$ we have $\phi(x) = a^j$, where $S^j \to a^j$ is the prefix rule with the longest length that matches x. We also refer to a classifier as an encoding.

To guarantee that the classification function of every encoding is well defined, we assume a default action $a^- \in \mathcal{A}$ to which headers that do not match any of the rules are mapped. We refer to a classifier that implements a function ϕ by C^ϕ.

Definition 5: We assume during the classification that packets appear according to a **header distribution** P, where p_x denotes the probability of a header x.

A classification function α and a header distribution P are the input of the problems discussed in this paper. The classification function can be described by a corresponding classifier C^α. In practice the exact header distribution might be unknown and instead it is only estimated by traffic sampling. Eventually, this estimation might result in some performance degradation, which is examined in Section VIII.

For a given classification function α we say that a classifier is an exact representation if it implements exactly the same function. A classifier is an exact representation of only one classification function. Note that the same classification function can be represented by several classifiers, possibly with different number of rules. As mentioned, an exact representation with the smallest number of prefix rules can be computed with the ORTC algorithm [16]. For a given α, we denote this minimal number of rules by n_0.

This paper studies a scenario in which the classification module can store fewer rules than the minimal number required for an exact representation. For instance, in the example from Table I, with fewer than four rules we cannot guarantee correct classification for all inputs.

III. Optimization Problems

A. Formal Definitions

We first define a metric that estimates the similarity of a classifier to a given classification function.

Definition 6: Let α be a classification function and P a header distribution. Let ϕ be a second classification function. The **correctness ratio** of a classifier that implements ϕ, denoted by $R_P(\alpha, \phi)$, is the probability of a header drawn according to the header distribution P to be classified to the same action in α and in ϕ. Formally,

$$R_P(\alpha, \phi) = \sum_{x \in \{0,1\}^W | \alpha(x) = \phi(x)} p_x. \tag{1}$$

In the first optimization problem, we would like to find an encoding with a limited number of rules that achieves a maximal correctness ratio.

Problem 1 (Approximate Classification): For a given classification function α, a header distribution P and a given number of prefix rules n, find a classifier C^ϕ with at most n rules that obtains a maximal correctness ratio

$$\mathcal{G}(n, \alpha, P) = \max_{C^\phi, |C^\phi| \leq n} R_P(\alpha, \phi). \tag{2}$$

Note that in the Approximate Classification problem a header that is not classified correctly can be mapped to an arbitrary action.

We denote by $\mathcal{G}(n, \alpha, P)$ the optimal (maximal) value of the above function. We refer to a legal encoding (with at most n rules) that obtains this correctness ratio as an approximation-optimal encoding. We also define the **error ratio** of a classifier that implements ϕ as $1 - R_P(\alpha, \phi)$.

In typical network applications, incorrect classification can be harmful. It can be more useful to avoid any wrongly classified packets by leaving some packets unclassified. We define an indication for headers that cannot be classified correctly by a given encoding. We denote this unique action by '?' and by $\mathcal{A}^*$ the generalized set of actions $\mathcal{A}^* = \mathcal{A} \cup \{\text{'?'}\}$. We would look for a classifier that for every header either returns the correct classification value or the indication '?'. Upon receiving such an indication, the classification can be completed, e.g. by using a slower traditional module, in mechanism similar to memory caching.

We can now define the main optimization problem, where we look for an encoding that obtains a maximal correctness ratio while returning the action of '?' for all headers that are not classified correctly.

Problem 2 (Cached Classification): For a given classification function α, a header distribution P and a given number of prefix rules n, find a classifier C^ϕ with at most n rules that obtains a maximal correctness ratio while satisfying $\forall x \in \{0,1\}^W$,

$$(\phi(x) = \alpha(x)) \text{ or } (\phi(x) = \text{'?'}). \tag{3}$$

We denote by $\mathcal{H}(n, \alpha, P)$ the optimal (maximal) correctness ratio that can be obtained while satisfying this additional condition and we say that an encoding that achieves this is a cache-optimal encoding. In the context of this problem, we define the quantity $1 - R_P(\alpha, \phi)$ as the **cache miss ratio**. This is the fraction of headers mapped to '?'.

Ways for coping with incorrect classifications and non-classified traffic have been mentioned in Section I. Note that for both problems the rules in an optimal encoding are not necessarily a subset of the rules in an exact classifier enabling to achieve even higher correctness ratios.

B. A Real-Life Illustrative Example

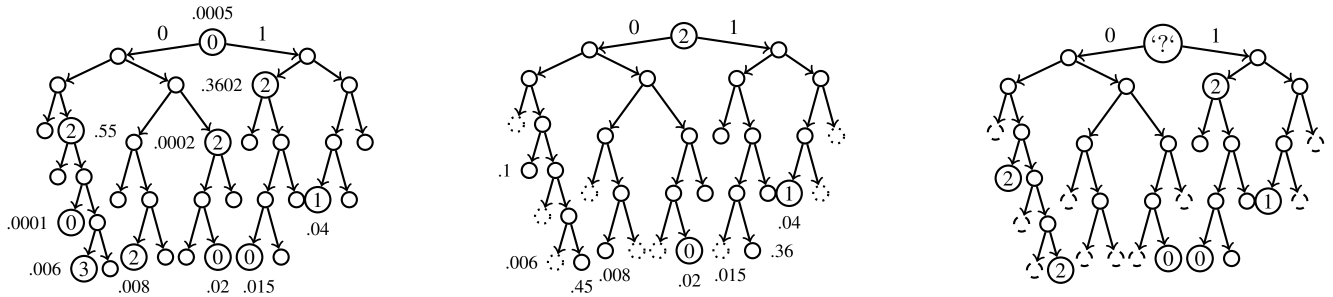
A classifier with prefix rules is often represented as a labeled binary prefix tree. Each node of the tree corresponds to a prefix rule, given by the transition bits along the path from the root node. A node that corresponds to a rule in the classification is labeled with the rule action. Fig. 2 shows an example of a binary tree which is a small branch of the tree for a real classifier. The left arc corresponds to a 0 bit transition, and the right to a bit of 1. Fig. 2(a) shows the ORTC compressed prefix tree. Here, ORTC requires 10 rules in total and guarantees 100% classification. The actions appear with labels in the tree and the probability of a header to have a longest match in a rule according to a real-life trace is illustrated. These probabilities rely on the header distribution of the captured traffic traces.

In Fig. 2(b) the probability that a given header is included within a given subtree appears next to the subtree such that in each subtree all headers have a match in the same rule. Subtrees without such numbers have a negligible probability. The result shown for optimal Approximate Classification with 3 rules can classify correctly 97.84% of the packets, and obtains false classifications for the rest. The subtrees from which headers are not classified correctly are drawn with dotted circles. The corresponding encoding has the prefix rules $01101/5 \to 0$, $1100/4 \to 1$ and $-/0 \to 2$. Fig. 2(c) shows the results of optimal Cached Classification with 7 rules. It can correctly classify 98.52% of the packets. It returns '?' for headers that cannot be classified correctly and there is no false classification. See also Table II in Sec. VIII how the correctness ratio increases by allowing more rules.

IV. Properties

Before solving these two problems, we would like to discuss some of their properties. First, to better understand the expected performance of solutions for the problems we discuss the values of the optimal correctness ratio in each of the problems.

The first observation compares the optimal values of the correctness ratios of the two problems for the same number of



(a) The minimal exact classifier requires 10 rules computed by ORTC [16]. The rule popularities are also drawn.

(b) Optimal *Approximate Classification* given 3 rules, $\mathcal{G}(3, \alpha, P) = 97.84\%$. The leaf probabilities are drawn (with negligible probabilities of the leaves without values). Headers in dotted leaves are incorrectly classified.

(c) Optimal *Cached Classification* given 7 rules, $\mathcal{H}(7, \alpha, P) = 98.52\%$. Headers matching the dashed leaves cannot be classified.

Fig. 2: An illustrative example of a branch of a real classifier. The action of a prefix rule is drawn as a node label.

rules. Intuitively, the requirement in the Cached Classification problem for a special indication on the headers that cannot be classified correctly results in a lower optimal ratio than that of the Approximate Classification problem.

Observation 1: For (n, α, P), the optimal correctness ratio of the Approximate Classification problem equals at least the optimal correctness ratio of the Cached Classification problem and the ratios are equal only if an exact representation exists. I.e.,

$$\mathcal{H}(n, \alpha, P) < \mathcal{G}(n, \alpha, P) < 1 \text{ or}$$
$$\mathcal{H}(n, \alpha, P) = \mathcal{G}(n, \alpha, P) = 1.$$

Indeed, in some cases $\mathcal{G}(n, \alpha, P)$ can be much larger than $\mathcal{H}(n, \alpha, P)$. Consider for instance a classifier (with a classification function α) that maps a single header with an arbitrarily small but positive probability to the action 1 and all other headers to the action 2. Assume a default action that is not one of these two actions. For $n = 1$, an Approximate Classification encoding of $-/0 \to 2$ classifies almost all headers correctly and $\mathcal{G}(n = 1, \alpha, P)$ is close to 1. Any Cached Classification encoding with a single rule must be $-/0 \to$ '?'. This encoding has a correctness ratio of $\mathcal{H}(n = 1, \alpha, P) = 0$.

Before presenting more observations, we define the popularity of a prefix rule in a classifier.

Definition 7: The **prefix rule popularity** of a rule in a classifier C is defined as the sum of the probabilities of all headers that have a longest match with the rule. For a rule $S^j \to a^j$ we denote this popularity by p^j. Formally,

$$p^j = \sum_{x \in \{0,1\}^W | S^j \text{ LPM } x} p_x, \tag{4}$$

where p_x denotes the probability for a header x to appear.

See also Fig. 2(a) as an example of prefix rule popularities drawn next to the nodes along with the probabilities of each leaf drawn on Fig. 2(b).

We now present a general property of the classifiers. Intuitively, the contribution of a rule to the correctness ratio is limited by its popularity.

Lemma 1: Let C^ϕ be a classifier. By removing a rule $S^j \to a^j$ from the classifier the correctness ratio of the Approximate Classification problem is decreased by at most p^j, where p^j denotes the rule popularity according to Definition 7.

The next observation suggests a lower bound for the optimal correctness ratio for the first problem. The bound is obtained as the ratio achieved by a classifier with a subset of the rules in the input classifier. Without adding additional rules, a header with a longest match in the input classifier in a selected rule will have such a match also in the smaller classifier.

Theorem 1: Let $C^\psi = (S^1 \to a^1, ..., S^{n_0} \to a^{n_0})$ be a classifier with a minimal number n_0 of prefix rules for a given classification function α. Assume a header distribution P. Let p^i be the prefix rule popularity of $S^i \to a^i$ when the rules are ordered in a non-increasing order of their popularities. By relying on a classifier composed of the n rules with the largest popularities, the optimal correctness ratio satisfies

$$\mathcal{G}(n, \alpha, P) \geq \sum_{i \in [1,n]} p^i + 1 - \sum_{i \in [1,n_0]} p^i \geq n/n_0.$$

This intuitively leads to a simple greedy algorithm for Approximate Classification which selects the n rules with highest rule popularity (see also Table II(a) as an example).

We can also derive a lower bound for the optimal correctness ratio for the Cached Classification problem based on the ratio obtained by such a classifier.

Observation 2: Let n_0 denote the minimum number of rules needed for the exact classifier of classification function α. Let us sort the prefix rules according to a non-increasing order of their lengths. For $n \in [1, n_0]$, the optimal correctness ratio of the Cached Classification problem satisfies

$$\mathcal{H}(n, \alpha, P) \geq \Sigma_{j \in [1,n-1]} p^j.$$

Similarly this leads to a simple greedy algorithm for Cached Classification which selects the $n - 1$ longest rules and adds a last rule of $-/0 \to$ '?' (see also Table II(b) as an example).

V. Approximate Classification

In this section we present algorithms that obtain optimal solutions for the Approximate Classification problem as defined in Problem 1. We see the Approximate Classification scheme as a baseline scheme that can help with the understanding of the Cached Classification which is the main scheme of the paper. We first assume a simple case in which the given classifier is represented by rules with distinct actions and present an immediate solution for this case. Later, we describe a more general algorithm of a classifier with arbitrary actions. This algorithm relies on dynamic programming.

A. The Case of Distinct Actions

Assume that all rules of a classifier have distinct actions and that these actions differ from the default action a^-. Our first observation is that removing the j^{th} rule decreases the correctness ratio by exactly p^j; all traffic that previously matched this rule is now not classified correctly. Thus to maximize the correctness ratio, we should include the rules with the highest popularity.

Observation 3: Let C be a classifier with n_0 rules with distinct actions that also differ from the default action a^-. For $n \in [1, n_0]$, an optimal correctness ratio for the Approximate Classification problem is composed of n rules with the highest popularity among the n_0.

B. Arbitrary Actions

We now describe a dynamic programming based algorithm to find an approximation optimal encoding, according to the header distribution P, for a classifier with arbitrary actions. Our solution calculates encodings for nodes in the tree such that each encoding comprises a limited number of rules and includes a specific last rule.

Let r be the root of the complete binary tree of 2^W leaves that includes all headers. For a node x (represented by a corresponding prefix) in the complete binary tree, we consider an encoding with a maximal number of rules $n \in \mathbb{N}^+$ satisfying that its last rule (among the n) is of the form $x \rightarrow a$ for an action $a \in \mathcal{A}$. We define the function $g(x, n, a)$ as the maximal ratio of headers from the subtree x that can be classified correctly by such an encoding. Let $\phi(x, n, a)$ be an encoding with the above properties that achieves this ratio.

The next lemma relates the optimal correctness ratio $\mathcal{G}(n, \alpha, P)$ to a value of the function $g(x, n, a)$. It relies on the value of the function for the root r with a specific number of rules and last action.

Lemma 2: The optimal correctness ratio satisfies $\mathcal{G}(n, \alpha, P) = g(r, n+1, a^-)$ where r is the root node and a^- is the default action. Likewise, an approximation-optimal encoding is given by the first n rules in $\phi(r, n+1, a^-)$.

We start by setting the values of $g(x, n, a)$ for a leaf (header) x assuming a classifier with a classification function α. Since there exists an encoding with n rules all of the form $x \rightarrow \alpha(x)$ we have

$$g(x, n, a) = p_x \quad \text{ for } n \geq 1 \text{ if } a = \alpha(x).$$

Recall that p_x is the probability of a header to have the value of x. In addition,

$$g(x, 1, a) = 0 \text{ for } a \neq \alpha(x) \text{ and,}$$

$$g(x, n, a) = p_x \text{ for } n \geq 2.$$

We can have an encoding with two rules $(x \rightarrow \alpha(x), x \rightarrow a)$ that classifies x correctly for any action a.

More generally, for a given classifier consider the first matching rule for the 2^W headers represented by leaves in the binary tree of size 2^W. Consider a recursive partition of the set of leaves into halves until each subset of consecutive leaves contains headers that have a first match in the same rule. This partition divides the headers into disjoint *monochromatic* subtrees. As explained in [30] the number of these monochromatic subtrees equals at most $W \cdot n_0$ where n_0 is the number of rules in the classifier.

Let y be a prefix that represents such a monochromatic subtree, i.e. a subtree in which all headers have the first matching in the same rule. Let a^y denote the action of that rule. The above formulas for a leaf can be generalized for such a subtree as follows.

$$g(y, n, a) = p_y \quad \text{ for } n \geq 1 \text{ if } a = a^y$$

where p_y is the probability of a header to be included in the subtree represented by y. Likewise,

$$g(y, 1, a) = 0 \text{ for } a \neq a^y \text{ and,}$$

$$g(y, n, a) = p_y \text{ for } n \geq 2.$$

The next lemma suggests a recursive formula for the value of $g(x, n, a)$ for a node x that is not a leaf. Similarly, the encoding $\phi(x, n, a)$ is calculated based on the two encodings for the left and right subtrees of x according to the value that is selected among the detailed cases.

Lemma 3: For a non-leaf node x and number of rules $n \geq 1$, the function $g(x, n, a)$ satisfies $g(x, n, a) = \max$

$$\left\{ \begin{array}{c} \max\limits_{m \in [1,n]} g(x_{\mathcal{L}}, m, a) + g(x_{\mathcal{R}}, n-m+1, a), \\ \max\limits_{m \in [1,n-1], a_1 \in \mathcal{A}} g(x_{\mathcal{L}}, m, a_1) + g(x_{\mathcal{R}}, n-m, a_1) \end{array} \right\},$$

where $x_{\mathcal{L}}$ and $x_{\mathcal{R}}$ are the left child and the right child of the node x.

The algorithm works as follows. Based on the rules in the given classifier, we divide the complete binary tree into monochromatic subtrees. After setting the function values for the corresponding nodes, we calculate based on the recursive formulas from Lemma 3 the function values and the corresponding encodings for internal nodes. An optimal encoding is obtained by Lemma 2 as the encoding for specific parameter values for the root of the complete binary tree.

Theorem 2: The algorithm obtains an optimal solution for the Approximate Classification problem.

Theorem 3 describes the time and space complexity of the algorithm. This analysis simply relies on the above description of the algorithm.

Theorem 3: The Approximate Classification problem can be optimally solved in $O(W \cdot n_0 \cdot |\mathcal{A}| \cdot n^2)$ time and $O(W^2 \cdot n_0 \cdot |\mathcal{A}| \cdot n^2)$ space, where n_0 is the number of rules in an exact encoding of the classifier.

The linear dependency of the time complexity and the quadratic dependency of the memory complexity in the header width W guarantee that the algorithm remains practical also for IPv6.

The dynamic-programming algorithm, calculates solutions for increasing tree sizes, has a general form similar to algorithms for calculating *semantically-equivalent representations* from [30] as well as [7] (in their one-dimensional case). These algorithms find optimal exact representations with a minimal number of prefix rules or with a minimal cost of prefix rules assuming a rule cost is determined by its action. Unlike the above, our algorithm also calculates, for trees given different restrictions on the rules number, best achievable solutions without completely correct classification. This enables finding the optimal solution for the problem first described here, that can often be *not semantically-equivalent*, and maximizes the traffic classified correctly while satisfying the constraint on the rule number.

VI. Cached Classification

In this section we present an algorithm that obtains an optimal solution for the Cached Classification problem as defined in Problem 2. This algorithm is also based on dynamic programming.

Recall that we define the generalized set of actions $\mathcal{A}^*$ as $\mathcal{A}^* = \mathcal{A} \cup \{`?`\}$. Here, we only consider encodings that for any header x either return the correct action $\alpha(x)$ or the action '?'. We define $h(x, n, a)$ as the maximal ratio of correctly classified headers in such an encoding with $n \in \mathbb{N}^+$ rules with a last rule of $x \to a$. In order to avoid illegal encodings, we use the function value of $-\infty$ if there does not exist an encoding that satisfies the above requirements. We also denote by $\psi(x, n, a)$ an example of an encoding that obtains this ratio.

As in the first problem we can deduce the optimal correctness ratio and an optimal encoding for the Cached Classification problem as follows.

Lemma 4: The optimal correctness ratio satisfies $\mathcal{H}(n, \alpha, P) = h(r, n + 1, a^-)$ where r is the root node and a^- is the default action. Likewise, an approximation-optimal encoding is given by the first n rules in $\psi(r, n + 1, a^-)$.

Again, let y be a monochromatic subtree that its headers, with a total probability of p_y, all have a first match in the same rule with an action a^y. For this problem, the encodings $(y \to a^y)$ and $(y \to `?`)$ are both legal but only the first of them classifies headers in y correctly. Accordingly,

$$h(y, 1, a) = p_y \text{ if } a = a^y \text{ and,}$$

$$h(y, 1, `?`) = 0.$$

On the contrary, an encoding of the form $(y \to a)$ is illegal if $a \neq a^y$ and $a \neq `?`$. Thus

$$h(y, 1, a) = -\infty \text{ if } a \notin \{a^y, `?`\}.$$

For any action $a \in \mathcal{A}^*$ the encoding $(y \to a^y, y \to a)$ is legal and classifies all headers in y correctly. Thus

$$h(y, n, a) = p_y \text{ for } n \geq 2, a \in \mathcal{A}^*.$$

For a non-leaf node x the values of $h(x, n, a)$ and the corresponding encoding $\psi(x, n, a)$ should be calculated recursively. Notice that if the two encodings for a left child $x_{\mathcal{L}}$ and a right child $x_{\mathcal{R}}$ are both legal, then the merged encoding for x is legal as well. Accordingly, the proof of the next lemma is similar to the proof of Lemma 3.

Lemma 5: For a non-leaf node x, and number of rules $n \geq 1$, the function $h(x, n, a)$ satisfies $h(x, n, a) = \max$

$$\left\{ \begin{array}{l} \max\limits_{m \in [1,n]} h(x_{\mathcal{L}}, m, a) + h(x_{\mathcal{R}}, n - m + 1, a), \\ \max\limits_{m \in [1,n-1], a_1 \in \mathcal{A}^*} h(x_{\mathcal{L}}, m, a_1) + h(x_{\mathcal{R}}, n - m, a_1) \end{array} \right\}.$$

With the described changes in the initial values of the function, the dynamic programming algorithm is the same as for Approximate Classification, and its optimality can be also deduced from the above discussion.

Theorem 4: The algorithm achieves an optimal solution for the Cached Classification problem.

The time and space complexity of the algorithm is essentially the same as described in Theorem 3. Putting together Lemma 4 and 5 we have the following theorem.

Theorem 5: The Cached Classification problem can be optimally solved in $O(W \cdot n_0 \cdot |\mathcal{A}| \cdot n^2)$ time and $O(W^2 \cdot n_0 \cdot |\mathcal{A}| \cdot n^2)$ space, where n_0 is the number of rules in an exact encoding of the classifier.

As stated in Observation 1, requiring an encoding for the Cached Classification problem to assign a special indication to every header that cannot be classified correctly has a cost of potential lower performance. In Section VIII we compare the optimal correctness ratios of the two problems.

VII. More General Classifiers

In this section, we generalize our novel approach of lossy compression to additional types of classifiers and allowed misclassifications.

A. Numerical Classification

We describe new metrics to capture the notion of similarity between classifiers. Then, we define new optimization problems for finding limited-size classifiers and explain how the previously mentioned algorithms can be modified to obtain optimal results for the new problems.

In these additional problems, we distinguish between two classifiers, even if they incorrectly classify the same set of headers, based on the exact values of the actions for the incorrectly classified headers. Assume a classifier in which the possible classification results (thus far called actions) are among a set of numerical values, i.e. $\mathcal{A} \subseteq \mathbb{R}$. Then, the difference $a_1 - a_2$ and the absolute difference $|a_1 - a_2|$ of two actions $a_1, a_2 \in \mathcal{A}$ are well defined. For instance, in such a numerical classification a header of a flow can be mapped to its required QoS level or to its allowed traffic rate.

The following problem generalizes the Approximate Classification problem. Here, we limit the encoder to classify a header to a value not smaller than the correct one. This can be useful for instance when a flow must be allocated at least its QoS level.

Problem 3 (One-Sided Approximate): For a classification function α, a header distribution P and a number of prefix rules n, find a classifier C^{ϕ} with at most n rules that obtains a maximal correctness ratio $R_P(\alpha, \phi)$ while satisfying $\forall x \in \{0,1\}^W$

$$\phi(x) \geq \alpha(x).$$

To solve Problem 3, we define $b(y, n, a)$ as the maximal correctness ratio in an encoding with n rules for headers in a subtree rooted by a node y such that the last rule is $y \to a$. Again, let p_y be the probability of a header to be included in the subtree represented by y and let $\mathcal{B}(n, \alpha, P)$ be the optimal value of the correctness ratio in this constrained problem. Our algorithm is based on the following lemma. The initial values of the function $b(y, n, a)$ for monochromatic trees enforce the restriction on the classification values for any header. The optimal correctness ratio again can be calculated based on the root node r.

Lemma 6: The function $b(y, n, a)$ satisfies

(i) For a monochromatic node y with a corresponding action a^y: $b(y, 1, a^y) = p_y$, $b(y, 1, a) = 0$ for $a > a^y$, $b(y, 1, a) = -\infty$ for $a < a^y$. Likewise, $b(y, n, a) = p_y$ for $n \geq 2, a \in \mathcal{A}$.

(ii) $\mathcal{B}(n, \alpha, P) = b(r, n+1, a^-)$ and for a non-leaf node y, $b(y, n, a) = \max$

$$\left\{ \begin{array}{l} \max\limits_{m \in [1,n]} b(y_{\mathcal{L}}, m, a) + b(y_{\mathcal{R}}, n-m+1, a), \\ \max\limits_{m \in [1,n-1], a_1 \in \mathcal{A}} b(y_{\mathcal{L}}, m, a_1) + b(y_{\mathcal{R}}, n-m, a_1) \end{array} \right\}.$$

In Problems 4 and 5, our goal is to find a classifier that minimizes the average difference between the requested actions and the obtained one. Given a classifier with a classification function α, we define the dissimilarity of a classifier with a classification function ϕ as $\Delta_P(\alpha, \phi) = \sum_{x \in \{0,1\}^W} p_x \cdot |\alpha(x) - \phi(x)|$. This for instance can represent a scenarios where we would like to match a flow a QoS level similar as possible to its required one while all kinds of errors are possible. While in the next problem, there are no constraints on the obtained actions for a specific header, in the later problem every header must be classified to a value not smaller than its corrected value.

Problem 4 (Unconstrained Dissimilarity): For a classification function α, a header distribution P and a given number of prefix rules n, find a classifier C^ϕ with at most n rules that minimizes the dissimilarity $\Delta_P(\alpha, \phi)$.

Problem 5 (One-Sided Dissimilarity): For a classification function α, a header distribution P and a given number of prefix rules n, find a classifier C^ϕ with at most n rules that minimizes the dissimilarity $\Delta_P(\alpha, \phi)$ while satisfying $\forall x \in \{0,1\}^W \ \phi(x) \geq \alpha(x)$.

Notice that Problem 4 and Problem 5 are minimization problems, unlike the previous problems.

We derive dynamic programming based solutions also for these problems. We define $\mathcal{U}(n, \alpha, P), \mathcal{O}(n, \alpha, P)$ as the optimal (minimal) dissimilarity values that can be obtained for the last two problems with n rules given α and P.

To solve Problem 4, we define $u(y, n, a)$ as the minimal possible dissimilarity value obtained in an encoding for headers in a subtree rooted by a node y with n rules such that the last rule is $y \to a$. Again, let p_y be the probability of a header to be included in the subtree represented by y.

Lemma 7: The function $u(y, n, a)$ satisfies

(i) For a monochromatic node y with a corresponding action a^y: $u(y, 1, a) = |a - a^y| \cdot p_y$ and $u(y, n, a) = 0$ for $n \geq 2$.

(ii) $\mathcal{U}(n, \alpha, P) = u(r, n+1, a^-)$ and for a non-leaf node y, $u(y, n, a) = \min$

$$\left\{ \begin{array}{l} \min\limits_{m \in [1,n]} u(y_{\mathcal{L}}, m, a) + u(y_{\mathcal{R}}, n-m+1, a), \\ \min\limits_{m \in [1,n-1], a_1 \in \mathcal{A}} u(y_{\mathcal{L}}, m, a_1) + u(y_{\mathcal{R}}, n-m, a_1) \end{array} \right\}.$$

Similarly we define the function $o(y, n, a)$ for Problem 5 and have the following.

Lemma 8: The function $o(y, n, a)$ satisfies

(i) For a monochromatic node y with a corresponding action a^y: If $a \geq a^y$ then $o(y, 1, a) = |a - a^y| \cdot p_y$ and if $a < a^y$ then $o(y, 1, a) = \infty$. In addition, $o(y, n, a) = 0$ for $n \geq 2$.

(ii) $\mathcal{O}(n, \alpha, P) = o(r, n+1, a^-)$ and for a non-leaf node y $o(y, n, a) = \min$

$$\left\{ \begin{array}{l} \min\limits_{m \in [1,n]} o(y_{\mathcal{L}}, m, a) + o(y_{\mathcal{R}}, n-m+1, a), \\ \min\limits_{m \in [1,n-1], a_1 \in \mathcal{A}} o(y_{\mathcal{L}}, m, a_1) + o(y_{\mathcal{R}}, n-m, a_1) \end{array} \right\}.$$

The dynamic programming algorithms can be easily derived from the above formulas. The analysis of their time and memory complexities is the same as for the previously mentioned algorithms.

B. Two-Dimensional Classifiers

We briefly discuss how all the presented algorithms can be generalized for two-dimensional prefix classifiers, a popular class of classifiers in which rules are defined on two fields such as the source IP and the destination IP addresses. A two-dimensional prefix rule is composed of two one-dimensional prefixes and allows headers that match in both fields. In order for the LPM-based matching to be unambiguous, we assume as in [30] that the given classifier is consistent. In such a classifier, any two rules are either disjoint or nested, i.e. either the set of matching headers in one rule is a subset of the set in the second or these sets are disjoint.

With a limited number of allowed rules, our goal is to find a classifier that achieves a maximal correctness ratio based on a known distribution of the two-dimensional headers. For a prefix x in the first field and a prefix y in the second, we calculate an optimal encoding of the headers in the rectangle (x, y). Such a rectangle represents the Cartesian product of the two subtrees that correspond to the prefixes x, y in the two fields. The algorithms for all discussed problems can be generalized in a similar way. The key observation is that an optimal encoding for the rectangle is obtained by splitting it into two halves along one of the fields. A similar claim appears in [30] regarding exact representations of classifiers.

C. Non-Prefix Classifiers

The discussed Approximate Classification and Cached Classification problems were defined in the context of limited-size classifiers that are restricted to include only prefix rules. The two problems can be easily generalized for maximizing the correctness ratio with encodings that are not necessarily prefix and the priority of rules is determined by their order. Such encodings can represent a more general class of classifiers that are not necessarily LPM-based. We refer to such not-necessarily prefix rules as general rules and denote these two generalized versions of the problems by the Generalized Approximate Classification and the Generalized Cached Classification problems. Clearly, since the generalized problems consider a superset of the possible encodings in the original problems, they can achieve at least the same correctness ratio and in some cases an improved ratio can be obtained. Since some classification modules, e.g. TCAM architectures are not constrained to include only prefix rules, solutions for the generalized problems might be useful. Unfortunately, we show that the two generalized problems are NP-hard. To show that we present a reduction from the problem of finding an exact representation of a classifier with a minimal number of general rules. This problem was proved to be NP-hard [31].

Theorem 6: The Generalized Approximate Classification and the Generalized Cached Classification problems are NP-hard.

Proof: We present a reduction from the mentioned above NP-hard problem. Consider a classifier with n_0 rules that defines a classification function α for which a representation with a minimal number of rules is required. Assume an arbitrary header distribution P for which all headers appear with a positive probability. For each $n \in [1, n_0]$, find a solution to the Generalized Approximate Classification problem that achieves a correctness ratio of 1. For relatively small values of n, such a solution might not exist. Such a solution always exists for $n = n_0$, i.e. $\mathcal{G}(n_0, \alpha, P) = 1$. Finally, return a solution obtained by the minimal n for which such a solution can be found. Such a solution is necessarily an encoding of the classifier with the minimal possible number of rules. Note that a linear number of values for n are examined (even a binary search can be performed here with examining linear number of values for n) and the hardness of the Generalized Approximate-Classification problem follows by this reduction. A similar solution can be obtained by solving the Generalized Cached-Classification problem while again requiring a correctness ratio of 1. Note that for both problems, if all headers appear with a positive probability, a solution that obtains a correctness ratio of 1 is necessarily an exact representation. ■

D. Supporting Updates

Classifiers have to support updates. These updates can include a change in the header distribution, an insertion of a new rule, a deletion of a rule, and a modification of the action in an existing rule. We discuss how to support these changes in the solutions for the two main optimization problems.

The suggested process for dealing with updates has two steps. The first step will be required *to keep the correctness of the encodings* and will be performed right after the update. The second step can *improve the performance of an encoding* (i.e., its correctness ratio) but is not required for coherency. This step can be run offline, either after a fixed number of changes or periodically in time. In general, supporting updates is easier in an encoding for the Approximate Classification problem (Problem 1) in comparison with an encoding for the Cached Classification problem (Problem 2). The reason is that for the first problem, any encoding with at most n rules is considered legal (although it might achieve a non-optimal correctness ratio due to the update). On the contrary, for the second problem any encoding must return for every header either its correct action or the unique action '?'. Thus a change in the required action even for a single header can make the encoding illegal.

A possible update that we would like to support is *a change in the header distribution*. Notice that according to the definition of Problem 2, the property of the returned action for a header is required for all headers, including those that with probability 0. This guarantees that a legal encoding for this second problem remains legal even after a change in the header distribution that includes increasing the probability of a header from 0 to a positive value. As mentioned, any solution is legal for the first problem before or after the change in header distribution. Such a change can significantly degrade the correctness ratio of both problems and running the dynamic-programming solution from the beginning is always possible. In some cases, we can save running time by relying on the solution earlier to the change with its calculations for some of the subtrees in the binary tree. In particular, if the probability is not changed for all headers within a subtree, all the calculations for the nodes it includes remain correct. Moreover, based on the formulas of the dynamic-programming if the probabilities of all headers in a subtree are changed by a fixed multiplicative factor (either smaller or greater than 1) there is no need to calculate the values and the corresponding encodings for all nodes in this subtree. Due to the linearity of the formulas in the elements' probabilities, we can simply update the values of the functions $g(x, n, a)$ or $h(x, n, a)$ (for a node x within the subtree) by multiplying them by the same factor while the encodings that obtain these values remain without a change.

Another kind of a possible update includes *a rule insertion, a rule deletion or an action update of an existing rule*. As mentioned, such a change can make an existing encoding for the second problem to be illegal when a header is classified to an incorrect action that is neither its action nor '?'. In general, a change in a rule can affect only headers that are within the subtree of the rule. Moreover, in the dynamic program this change influences at most $(W + 1)$ nodes in the path from the root of the whole binary tree to the node that represents the affected subtree. To deal with the change, we can also keep (especially for a large n) a small number of unused rules. Then, for correctness in any case of a rule change, we can temporarily add a rule with '?' for this subtree. Another option is that when a new rule is inserted, we can add it to the existing classifier as is if there are no some more specific rules or with '?' if this is not the case. If a rule is deleted and it appears in the encoding it can be changed to the action of the longest matching prefix for this rule among the other existing rules. For the first problem the coherency of the solution is kept even after such a change while this might influence the correctness ratio. In both problems, the decision whether or when the existing encoding should be updated can rely on the ratio of headers affected by this change.

A transition between two rule configurations is often required to be atomic, avoiding intermediate states combining the two configurations as supported by a feature of the OpenFlow specification [32]. There are several ways to implement such an atomic bundle change. This can be achieved through a double-buffered flow-table [33] maintaining an active and a shadow table such that the first table is used to serve the traffic when the other is updated before the two tables are swapped. Another recent approach [34] relies on adding to the rules a time field describing the time they apply. A query of a packet is processed by aligning the packet with the timestamp it entered the network. Rules of the both configurations can appear within the same time, such that the changed rules examine the timestamp field in order to apply only before or after some agreed transition time.

VIII. Experimental Results

We performed extensive simulations to validate the proposed approaches on a workstation with a 2.50GHz Intel Core i5 CPU. Since the effectiveness of the proposed approach depends on the correlation between the header distribution and the input classifiers, we have arranged a measurement in a campus network of the Budapest University of Technology and Economics (BME) between December 11-14, 2015, exporting the Forwarding Information Base (FIB) and capturing more than 2 billion packets to compute the prefix popularities. The measured link was a 10Gigabit Ethernet port of a Cisco 6500 Layer-3 switch, which transfers the traffic on a campus site to the core layer of the local network. We refer to the data from this measurement as the BME FIB and trace.

In addition to the BME FIB, we examined additional classifier instances of 6 access and core FIB instances, used previously in [17], as summarized in Table III. The higher order Entropy of the FIB is also given in KBytes describing the theoretical memory lower bound of lossless compression [17]. As a rough comparison we may treat a TCAM rule of 32 matching bits as 4 bytes of information. For the calculation of the prefix popularities, we also made used of a Yahoo's (G4) network flows dataset. The dataset includes almost a billion packets collected from three border routers connected to Yahoo data centers in October 11, 2007. All IP addresses in the dataset are anonymized using a random permutation algorithm. Fig. 3 shows the cumulative distribution function of the popularities of the /24 long prefixes in the BME and Yahoo trace. For example in the BME trace where a higher locality is observed, roughly 50% of the traffic is mapped to ten /24 prefixes. This illustrates how biased is the distribution faced in the classification process.

In total we solved 2×7 problem instances with the following six algorithms. We implemented the two dynamic programming schemes, providing optimal solutions for both the Approximate Classification and Cached Classification problems. We also implemented four other schemes whose solution to the real-life illustrative example of Fig. 2 in Sec. III-B is shown on Table II. Three of these schemes compute a subset of rules of the ORTC representation, which is an exact representation with minimal number of rules. For Cached Classification the last rule is always $-/0 \rightarrow$ '?'. To maintain the cache correctness, a prefix rule can be selected, only if every existing longer prefix intersecting rule is also selected. We implemented the greedy algorithm for Approximate Classification which selects the n rules with highest prefix rule popularity. We also implemented two greedy schemes for Cached Classification: the *Greedy Cashed Classifier* selects the longest rules and among the same length the ones with highest prefix rule popularity; the *Dep-Set Cache Classifier* [29] selects a subtree that has the highest popularity in proportion to the number of rules in the subtree. Finally, the *Pragmatic Cached Classifier* [25] selects leaf with the highest popularity. Table II shows for both schemes how the correctness ratio increases by allowing more rules in each of the four schemes.

The right side of Table III shows the number of rules required to reach 90%, 95%, 99%, 99.9% and 99.99% correctness ratio for the Approximate Classification and Cached Classification problems on the BME trace. For example, 107K rules are required for exact classification of the HBONE FIB with ORTC, and surprisingly with 59 rules a correctness ratio of 95% can be reached for the Approximate Classification, and 678 rules are needed for the Cached Classification. Table IV shows the same results using the prefix popularities computed according to the Yahoo trace. The differences are not significant. Roughly 2% of the rules required by ORTC was sufficient to classify 99% of the traffic by the fast classifier for Cached Classification. With 10K-20K rules we can achieve a very high 99.99% correctness ratio, which is still approximately 10% of the ORTC representation.

Fig. 4 shows the average error ratio for Cached and Approximate Classification over all the FIBs for the BME traces. It is an average of the 7 instances. The 95% confidence interval (among the instances) of the optimal algorithms is also plotted in one side as a shadow of the curves. Note that logarithmic scale is used on both axes. We plot $1 -$ correctness ratio, which is also called the *error ratio* or the *cache miss ratio* for the Approximate Classification or the Cached Classification problems, respectively. On average Cached Classification required 2.77 times more rules than Approximate Classification for the same correctness ratio. This factor decreases as we have a larger correctness ratio. The figure also shows the results of the four other schemes. For Approximate Classification the greedy algorithm performs closest to the optimal solution requiring 10%-30% more rules compared to the optimal solution. For Cached Classification the schemes that select a subset of the ORTC rules achieve bad performance. Note that the ORTC ruleset is the minimal exact representation. The Pragmatic Caching algorithm provides a decent performance, by dividing the tree into disjoint subtrees and selecting those with highest popularity. This approach is close to the optimal if the number of rules is small. While if the fast classifier has more than 1000 rules the difference becomes large. See also Table IV for the comparison of these four methods.

Finally, Fig. 5 shows the estimated increase in throughput compared to the exact classifier with respect to the size of the fast classifier. We compare the performance of the optimal Cached Classification and the Pragmatic Caching with the BME and Yahoo traces using the HBONE FIB. We assume that the throughput of the fast classifier is roughly 5 times that of the slow classifiers in inverse proportion to their latencies [29]. In case of cache miss the lookup is performed on both the fast and exact classifier which slightly reduces the total throughput. Better performance is observed for the BME trace

TABLE II: Illustration of the greedy algorithms for the two schemes on the example of Fig. 2: rules are considered in different orders based on their popularities and lengths. For a number of rules n the obtained greedy correctness ratio is compared with the optimal ratio.

(a) Approximate Classification

n	$\mathcal{G}$ opt.	Greedy Approx Class. ratio	rule	pop.
1	.9184	.55	$001/3 \to 2$	.55
2	.9584	.9102	$10/2 \to 2$	.3602
3	.9784	.9502	$1100/4 \to 1$	.04
4	.9934	.9702	$01101/5 \to 0$	.02
5	.9994	.9852	$10100/5 \to 0$	.015
6	.9996	.9932	$01010/5 \to 2$	.008
7	.9997	.9992	$001110/6 \to 3$	.006
8	.9998	.9997	$-/0 \to 0$	.0005
9	.9999	.9999	$011/3 \to 2$	.0002
10	1	1	$00110/5 \to 0$	.0001

(b) Cached Classification

n	$\mathcal{H}$	Greedy Cached Class.		Dep-Set Cache Class. [29]	
	opt.	rule	ratio	rule	ratio
2	.45	$001110/6 \to 3$	.006	$10100/5 \to 0$	
3	.81	$01101/5 \to 0$	.026	$10/2 \to 2$	.3752
4	.91	$10100/5 \to 0$	.041	$00110/5 \to 0$	
5	.95	$01010/5 \to 2$	.049	$001110/6 \to 3$	
6	.97	$00110/5 \to 0$	.0491	$001/3 \to 2$	.9313
7	.9852	$1100/4 \to 1$	.0891	$1100/4 \to 1$	.9713
8	.9932	$001/3 \to 2$	.6391	$01101/5 \to 0$	.9913
9	.9993	$011/3 \to 2$	.6393	$01010/5 \to 2$	.9993
10	1	$10/2 \to 2$	.9995	$011/3 \to 2$	
11		$-/0 \to 0$	1	$-/0 \to 0$	1
		with a last rule $-/0 \to$ '?'			

n	Pragmatic rule caching [25] rule	ratio
2	$001111/6 \to 2$	0.45
3	$10101/5 \to 2$	0.81
4	$0010/4 \to 2$	0.91
5	$1100/4 \to 1$	0.95
6	$01101/5 \to 0$	0.97
7	$10100/5 \to 0$	0.985
8	$01010/5 \to 2$	0.993
9	$001110/6 \to 3$	0.999
10	$0111/4 \to 2$	0.9991
11	$01100/5 \to 2$	0.9992
12	$01011/5 \to 0$	0.9993
13	$100/3 \to 2$	0.9994
14	$0100/4 \to 0$	0.9995
15	$1011/4 \to 2$	0.9996
16	$1101/4 \to 0$	0.9997
17	$00110/5 \to 0$	0.9998
18	$111/3 \to 0$	0.9999
19	$000/3 \to 0$	1
	with a last rule $-/0 \to$ '?'	

TABLE III: Description of FIBs examined with the number of distinct actions, the number of leaves and nodes in their original representations, and the number of ORTC rules. The required number of rules in the Cached and Approximate classifier are described for different correctness ratio. Results are based on the the BME trace.

FIB Name	#leaves	#nodes	ORTC (#rules)	Entropy (KB)	Optimal Approximate Classifier (#rules) 90%	95%	99%	99.9%	99.99%	Optimal Cached Classifier (#rules) 90%	95%	99%	99.9%	99.99%
BME (TAZ)	150095	300189	49285	56	4	15	157	778	2184	176	444	1513	3460	6160
SFR-HMS	235624	471247	71802	90	2	17	231	1161	3613	205	560	2048	5582	10679
AS1221	261889	523777	94231	115	10	45	405	1808	5090	223	601	2294	6455	12718
AS4637	105234	210467	35872	41	4	10	128	662	1773	86	236	1014	2463	4316
AS6447	375261	750521	160835	277	52	172	1106	4245	11614	257	745	3103	10582	23628
AS6730	336828	673655	140481	209	35	126	848	3415	9426	254	702	2829	9083	19341
HBONE	284716	569431	107739	142	15	59	552	2336	6505	241	678	2720	8358	17682

TABLE IV: The required number of rules in the Cached and Approximate classifier are described for different correctness ratio. Results are based on the Yahoo trace.

FIB Name	Optimal Approximate Classifier 95%	99%	99.9%	99.99%	Greedy Approximate Classifier 95%	99%	99.9%	99.99%	Optimal Cached Classifier 90%	95%	99%	99.9%	99.99%	Pragmatic Cached Classifier 90%	95%	99%	99.9%	99.99%
SFR-HMS	44	436	1728	3882	70	547	2043	4418	392	701	2126	5145	7439	658	1379	4413	12338	20185
AS1221	260	984	2885	5896	339	1128	3259	6567	667	1222	2977	6619	9986	918	1824	5293	14104	23207
AS4637	68	423	1041	1962	99	481	1155	2157	325	574	1172	2279	3373	457	965	2299	5170	8439
AS6447	632	1979	5960	11157	755	2289	6917	12814	881	1664	4630	11794	18281	1102	2217	6986	19445	31654
AS6730	555	1798	5223	9871	667	2049	6037	11232	780	1493	4130	10083	15620	1032	2078	6482	17792	29013
HBONE	309	1119	3322	6820	462	1500	4529	8941	675	1281	3432	8778	14102	890	1804	5299	14680	24414
BME (TAZ)	76	448	1184	2467	103	495	1311	2730	443	755	1528	3030	4623	635	1274	3169	7715	12849

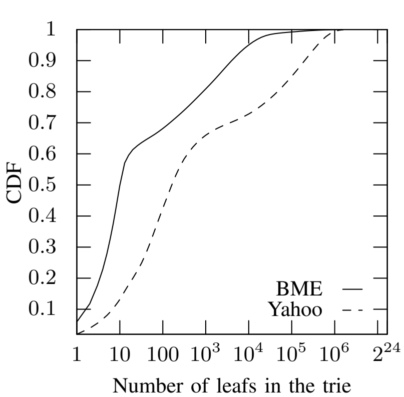


Fig. 3: The cumulative distribution function of the popularities of the /24 the IPv4 address space.

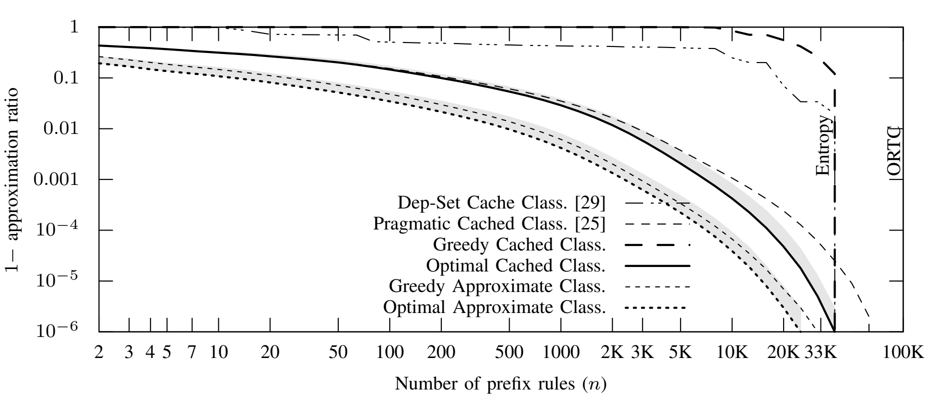


Fig. 4: The error ratio and the cache miss ratio vs. the number of rules. The average over the 7 FIBs with the BME trace. The 95% confidence interval is also plotted for the dynamic program. Note that the optimal exact representations (ORTC) [16] have 94320 rules in average, while the theoretical lower bound assuming 32 bits per rule would be 33214 rules [17].

that has higher locality. For instance, with 1000 rules with the BME trace the throughput increase is 4.37 and 4.25 for Cached Classification and Pragmatic Caching, respectively. For the Yahoo trace, the corresponding values are 3.93 and 3.54. In general the optimal Cached Classification scheme achieves up to 2.8% and 11% larger increase in the throughput for the BME and Yahoo traces in comparison with the Pragmatic Caching.

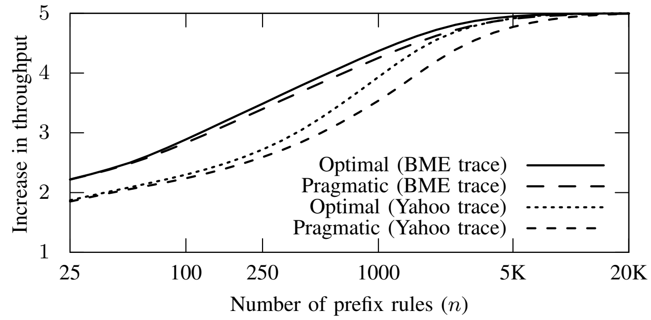


Fig. 5: The throughput increase compared to exact classifier vs. the size of the fast classifier for the HBONE FIB with Cached Classification and Pragmatic Caching. We assumed the throughput of the fast classifier is 5 times that of the exact classifier.

IX. Related Work

Reducing the TCAM size is a central idea of building energy efficient routers [35]–[38]. Compared to our concept the fundamental difference is that the power consumption is reduced because of selectively activating only a portion of the TCAM during lookup. The general idea is to implement the classification process in two steps, called 2-level architecture, the first step decides which TCAM segment to perform the lookup search. The first step can be either implemented in SRAM [36], [38], or with a TCAM table, called index TCAM, [35], [37]. It was demonstrated that these techniques can greatly reduce power, and improve the lookup time while supporting batch updates. In this paper we take an orthogonal approach and calculating an approximated classification in the first step, using a fast classifier.

The concept of using traditional and fast classifiers for rule caching is not new [26], [27]. In an LPM-based classification, the existence of a matching rule for an incoming packet among the set of cached rules does not necessarily mean that this is the correct rule, since there might exist a longer matching rule among the non-cached rules. [24] described schemes for selecting a subset of rules that avoids this phenomenon, known as the cache hiding problem. [39] tackled the same problem by introducing rule caching for fast line cards wakeup. [29] described a system that caches the most popular rules in a small TCAM while handling the others in software. A recent approach suggested distributing the rules of a classifier among several limited-size TCAMs in multiple locations [40], [41].

Compression of packet classifiers as well as of Forwarding Information Bases (FIBs) is another effective strategy to reduce TCAM power consumption. The problem was considered for a wide range of memories. The ORTC algorithm [16] obtains an optimal representation of a longest prefix match (LPM) classifier in the minimal possible number of prefix rules. A similar approach called FIB aggregation [42] suggests aggregating rules with the same action. Similar techniques are described in [43], [44]. Entropy bounds on the size of an LPM-based classifier and algorithms to obtain them were presented in [17]. [13] discussed how to reduce the width of a classifier by eliminating some of its fields. Codes for fixed-width memories have been described in [45], [46]. In particular, the problem of dealing with the limited size of TCAMs has been well studied. A wide range of memory-efficient representations of classifiers in TCAMs have been suggested [7], [47], [48]. For instance, the compression can be achieved by eliminating redundant rules [12], by learning the interactions between different rules [13], [14], or by performing block permutation [15]. Recently, a novel scheme took advantage of the IP address allocation to reduce memory required to represent network policies [49].

X. Discussion on the Applicability of the Results

Generality of the solutions: We have described algorithms that obtain the optimal classifier with a limited number of rules for the Approximate Classification problem, the Cached Classification problem, and for additional problems related to numerical classifiers. The algorithms differ in the initial values of the recursive formulas for the monochromatic subtrees. In all problems, the function value of a node for a possible limited-size encoding is given by the sum of the values of its two subtrees. As mentioned, the number of rules in an encoding achieved by combining two encodings for two adjacent subtrees does not depend on a specific metric. Let the function $I(\cdot)$ be the indicator function that takes the value of 1 if the condition that it receives as an argument is satisfied, and 0 otherwise. The algorithms can be generalized to any metric in which the value of an encoding that implements a function ϕ is given by $\sum_{x\in\{0,1\}^W} \mathcal{F}(\alpha(x), \phi(x), p_x)$ for an arbitrary function $\mathcal{F}$, and the constraints of the function ϕ can be also expressed based on the values for each header. In such cases, the similarity of the functions is separable for the different headers. In particular, for the described problems we have for Approximate Classification $\mathcal{F}(\alpha(x), \phi(x), p_x) = p_x \cdot I(\alpha(x) = \phi(x))$, for Cached Classification it can be $p_x \cdot I(\alpha(x) = \phi(x)) - 1 \cdot I(\phi(x) \notin \{\alpha(x), \text{‘?‘}\})$. Here, an incorrect classification of a single header decreases the function value for the complete binary tree by at least one and guarantees that its value will not be positive.

Dealing with attacks: A Cached Classifier can be affected by malicious traffic leading to performance degradation. This can be expressed in various aspects. First, malicious traffic can be used to pollute the cache content by artificially changing traffic distribution to reduce correctness ratio. Second, malicious traffic can try to achieve cache misses to heavily increase the load of the traditional exact classifier. We rely on [25] that recently studied the vulnerability of rule caching schemes to such attacks. This study explains that the influence of the first aspect is minor if exists due to the biased distribution of the real traffic and the popularity of the requested cached rules. It is claimed that the traffic rate required to avoid caching the significant and helpful rules is much larger than any practical rate. Accordingly, this limited effect on the traffic distribution guarantees that the efficiency of the approximate classification and the cached classification schemes will not be highly degraded by the malicious traffic. Both schemes optimally maximize the correctness ratio for their input distribution, a distribution that is very close to the real traffic distribution without the influence of the malicious traffic.

Regarding the second aspect, clearly the traffic that is mostly affected is the malicious traffic observing a larger delay; although, this can lead to unrequested large power consumption. Moreover, there are simple solutions to keep the cache content clean from attacks. The prefix popularity can be computed combining long term statistics with short term measurements. When some information of this traffic is available, we can also try to limit the influence of a single or a small amount of flows on the header distribution. For example, such an attack can be identified by common tools for detecting unrequested large amount of traffic such as tools for diagnosing DDoS attacks. We leave such identification of malicious traffic for future work.

XI. Conclusions

In this paper, we investigate algorithmic aspects of a limited-size and power efficient packet classifiers. In particular, we have described a lossy compression approach for limited-size classification modules. We have presented different similarity metrics for classifiers and developed algorithms that find optimal classifiers under various constraints. In particular, we have presented a scheme in which a special indication is always returned for headers that cannot be classified correctly. Then, a correct classification can be achieved by accessing the network controller or another memory level. We have explained how the approach can be applied to a wide range of classifiers within different modules. Extensive experiments showed a significant reduction in the size of real classifiers based on real traffic. According to our conservative estimations, 1-3% of the original TCAM size should be enough for correctly classifying 99% of the traffic. While we can show the extending the lossy compression methodology for classifiers with general, non-prefix rules can result in NP-hard problems, our future work includes developing techniques also for these scenarios.

References

[1] O. Rottenstreich and J. Tapolcai, "Lossy compression of packet classifiers," in *ACM/IEEE ANCS*, 2015.

[2] I. Gashinsky, "Datacenter scalability panel," in *North American Network Operators Group, NANOG 52*, 2011.

[3] D. Meyer, L. Zhang, and K. Fall, "Report from the IAB Workshop on Routing and Addressing," in *RFC 4984, IETF*, 2007.

[4] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, 2006.

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

[6] P. Bosshart, G. Gibb, H. Kim, G. Varghese, N. McKeown, M. Izzard, F. A. Mujica, and M. Horowitz, "Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN," in *ACM SIGCOMM*, 2013.

[7] A. X. Liu, C. R. Meiners, and E. Torng, "TCAM Razor: a systematic approach towards minimizing packet classifiers in TCAMs," *IEEE/ACM Trans. Netw.*, vol. 18, 2010.

[8] N. Kang, M. Ghobadi, J. Reumann, A. Shraer, and J. Rexford, "Efficient traffic splitting on commodity switches," in *ACM CoNEXT*, 2015.

[9] R. Ozdag, "Intel®Ethernet Switch FM6000 Series-Software Defined Networking," *Intel Coroporation*, 2012.

[10] D. Huffman, "A method for the construction of minimum redundancy codes," *Proc. IRE*, vol. 40, 1952.

[11] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Information Theory*, vol. 24, 1978.

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

[13] K. Kogan, S. I. Nikolenko, O. Rottenstreich, W. Culhane, and P. Eugster, "Exploiting order independence for scalable and expressive packet classification," *IEEE/ACM Trans. Netw.*, vol. 24, pp. 1251–1264, 2016.

[14] E. Norige, A. X. Liu, and E. Torng, "A ternary unification framework for optimizing TCAM-based packet classification systems," in *ACM/IEEE ANCS*, 2013.

[15] R. Wei, Y. Xu, and H. J. Chao, "Block permutations in boolean space to minimize TCAM for packet classification," in *IEEE Infocom MC*, 2012.

[16] R. Draves, C. King, S. Venkatachary, and B. Zill, "Constructing optimal IP routing tables," in *IEEE Infocom*, 1999.

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

[18] G. K. Wallace, "The JPEG still picture compression standard," *Commun. ACM*, vol. 34, 1991.

[19] D. L. Gall, "MPEG: A video compression standard for multimedia applications," *Commun. ACM*, vol. 34, 1991.

[20] M. Nilsson, *The audio/Mpeg Media Type*. RFC 3003, 2000.

[21] N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang, "Leveraging Zipf's law for traffic offloading," *Computer Communication Review*, vol. 42, 2012.

[22] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and S. C. Diot, "Packet-level traffic measurements from the Sprint IP backbone," *IEEE Network*, vol. 17, 2003.

[23] M. Fomenkov, K. Keys, D. Moore, and K. Claffy, "Longitudinal study of Internet traffic in 1998-2003," in *WISICT*, 2004.

[24] Y. Liu, V. Lehman, and L. Wang, "Efficient FIB caching using minimal non-overlapping prefixes," *Computer Networks*, vol. 83, 2015.

[25] K. Gadkari, M. L. Weikum, D. Massey, and C. Papadopoulos, "Pragmatic router FIB caching," in *IFIP Networking*, 2015.

[26] H. Liu, "Routing prefix caching in network processor design," in *ICCCN*, 2001.

[27] B. Talbot, T. Sherwood, and B. Lin, "IP caching for terabit speed routers," in *IEEE Globecom*, 1999.

[28] C. Kim, M. Caesar, A. Gerber, and J. Rexford, "Revisiting route caching: The world should be flat," in *Passive and Active Network Measurement (PAM)*, 2009.

[29] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Cacheflow: Dependency-aware rule-caching for software-defined networks," in *ACM Symposium on SDN Research (SOSR)*, 2016.

[30] S. Suri, T. Sandholm, and P. R. Warkhede, "Compressing two-dimensional routing tables," *Algorithmica*, vol. 35, 2003.

[31] R. McGeer and P. Yalagandula, "Minimizing rulesets for TCAM implementation," in *IEEE Infocom*, 2009.

[32] Open Networking Foundation, "Openflow switch specification," *Version 1.5.0*, 2014.

[33] J. H. Han, P. Mundkur, C. Rotsos, G. Antichi, N. H. Dave, A. W. Moore, and P. G. Neumann, "Blueswitch: Enabling provably consistent configuration of network switches," in *ACM/IEEE ANCS*, 2015.

[34] T. Mizrahi, O. Rottenstreich, and Y. Moses, "TimeFlip: Using timestamp-based TCAM ranges to accurately schedule network updates," *IEEE/ACM Trans. Netw.*, 2017.

[35] F. Zane, G. Narlikar, and A. Basu, "Coolcams: Power-efficient TCAMs for forwarding engines," in *IEEE Infocom*, 2003.

[36] W. Lu and S. Sahni, "Low-power TCAMs for very large forwarding tables," *IEEE/ACM Trans. Networking*, vol. 18, 2010.

[37] C. R. Meiners, A. X. Liu, E. Torng, and J. Patel, "Split: Optimizing space, power, and throughput for TCAM-based classification," in *ACM/IEEE ANCS*, 2011.

[38] T. Banerjee, S. Sahni, and G. Seetharaman, "PC-TRIO: A power efficient TCAM architecture for packet classifiers," *IEEE Trans. Computers*, vol. 64, 2015.

[39] T. Pan, T. Zhang, J. Shi, Y. Li, L. Jin, F. Li, J. Yang, B. Zhang, and B. Liu, "Towards zero-time wakeup of line cards in power-aware routers," in *IEEE Infocom*, 2014.

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

[41] K. Kogan, S. I. Nikolenko, P. Eugster, A. Shalimov, and O. Rottenstreich, "FIB Efficiency in Distributed Platforms," in *IEEE ICNP*, 2016.

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

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

[44] Y. Liu, X. Zhao, K. Nam, L. Wang, and B. Zhang, "Incremental forwarding table aggregation," in *IEEE Globecom*, 2010.

[45] O. Rottenstreich, M. Radan, Y. Cassuto, I. Keslassy, C. Arad, T. Mizrahi, Y. Revah, and A. Hassidim, "Compressing forwarding tables for datacenter scalability," *IEEE JSAC*, vol. 32, pp. 138–151, 2014.

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

[47] O. Rottenstreich, R. Cohen, D. Raz, and I. Keslassy, "Exact worst case TCAM rule expansion," *IEEE Trans. Computers*, vol. 62, pp. 1127–1140, 2013.

[48] O. Rottenstreich, I. Keslassy, A. Hassidim, H. Kaplan, and E. Porat, "Optimal In/Out TCAM encodings of ranges," *IEEE/ACM Trans. Netw.*, vol. 24, pp. 555–568, 2016.

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

APPENDIX

Proof of Observation 1: A representation that classifies all traffic correctly obtains a correctness ratio of 1 for both problems. If such a representation does not exist, consider an optimal encoding for the second problem. It must include at least one rule with '?' that matches at least one header. Based on this encoding, we can simply obtain a solution for the first problem that achieves a larger correctness ratio. To do so, we replace the action of '?' in such a rule by an action in $\mathcal{A}$ that corresponds to one of the headers with a first match in that rule. Such a change does not affect any header that was originally classified correctly, while at least one header previously mapped to '?' is now classified to its required action. Note that for the Approximate Classification a correctness ratio of 1 can be obtained also when some headers are not classified correctly as long as they have a probability of zero to appear. $\square$

Proof of Lemma 1: Any header that had a longest match in one of the rules other than $S^j \to a^j$ will have a longest match in the same rule after removing rule $S^j \to a^j$. $\square$

Proof of Theorem 1: In such a classifier, a header that was classified correctly having a longest match in a selected rule will have again a longest match in the same rule, being classified correctly. In addition, all headers that had no match in any of the n_0 will not match the subset of rules. Such headers are mapped to the default action in both cases. The last bound is deduced by a simple bound on the average of the largest n popularities and the consideration of the probability that a header does not match any rule. $\square$

Proof of Observation 2: Consider an encoding with n rules $S_1 \to a_1, \ldots, S_{n-1} \to a_{n-1}, -/0 \to$ '?' composed of the $n-1$ longest rules in the encoding of C^α and a last default rule that returns '?'. It classifies correctly all headers matching one of these $n-1$ longest rules in the exact encoding with n_0 rules and therefore achieves a correctness ratio of $\Sigma_{j \in [1,n-1]} p^j$. This encoding is legal since it returns '?' for any other header. $\square$

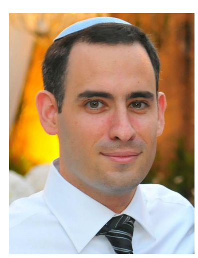
Proof of Lemma 2: Consider an encoding $\phi(r, n+1, a^-)$ that obtains the correctness ratio $g(r, n+1, a^-)$. In such an encoding the last rule of the form $r \to a^-$ is redundant since a^- is the default action. By eliminating this rule we can have an encoding of n rules that achieves the same ratio and therefore $\mathcal{G}(n, \alpha, P) \geq g(r, n+1, a^-)$. Likewise, for any encoding with n rules that obtains $\mathcal{G}(n, \alpha, P)$ we can add a rule of the form $r \to a^-$ while still obtaining the same correctness ratio. This is a legal encoding for $g(r, n+1, a^-)$. Thus we have $\mathcal{G}(n, \alpha, P) \leq g(r, n+1, a^-)$ and the equality is satisfied. $\square$

Proof of Theorem 3: As mentioned, the number of the monochromatic nodes is at most $W \cdot n_0$. The calculation of the trees is performed by a single processing of the rules in the input classifier. A binary tree has the property that the number of internal nodes is not greater than the number of leaves. Accordingly the total number of considered nodes is $O(W \cdot n_0)$. For each node and a given value of n, we consider n options for a specific value of a and another $O(n \cdot |\mathcal{A}|)$ options that refer to all values of a. Thus the total time complexity is $O(W \cdot n_0 \cdot |\mathcal{A}| \cdot n^2)$. Likewise, for each calculation we should keep an encoding of size $n \cdot W$. This result in a total memory complexity of $O(W^2 \cdot n_0 \cdot |\mathcal{A}| \cdot n^2)$. $\square$

Proof of Lemma 6: For a monochromatic node y an encoding $y \to a$ is legal if $a \geq a^y$ and achieves a positive correctness ratio only if $a = a^y$. Likewise, a legal encoding for a non-leaf node y is given by the merging of legal encodings for the two subtrees regardless of the specific optimization function. $\square$

Proof of Lemma 7: Here, for a monochromatic node y the encoding $y \to a^y$ with a single rule has a dissimilarity of 0, while an encoding of the form $y \to a$ for $a \neq a^y$ has a dissimilarity of $|a - a^y| \cdot p_y$. $\square$

Proof of Lemma 8: The proof is similar to the previous proofs. To avoid an illegal encoding of the form $y \to a$ for the monochromatic node y when $a < a^y$, we set the value of the function $o(y, 1, a)$ to be ∞. $\square$



Ori Rottenstreich is a Postdoctoral Research Fellow at the department of Computer Science, Princeton university, working with Prof. Jennifer Rexford. He received the B.S. in Computer Engineering (summa cum laude) and Ph.D. degree from the Electrical Engineering department of the Technion, Haifa, Israel in 2008 and 2014, respectively. He is a recipient of the Rothschild Yad-Hanadiv postdoctoral fellowship, the Google Europe PhD 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.

János Tapolcai received his M.Sc. ('00 in Technical Informatics), Ph.D. ('05 in Computer Science) degrees from Budapest University of Technology and Economics (BME), Budapest, and D.Sc. ('13 in Engineering Science) from Hungarian Academy of Sciences (MTA). Currently he is a Full Professor at the Department of Telecommunications and Media Informatics at BME. He is an author of over 110 scientific publications, he is the recipient of the Google Faculty Award and Best Paper Award in ICC'06, in DRCN'11, in NaNa'16. He is a TPC member of leading conferences such as IEEE INFOCOM (2012 - 2017), and is a winner of MTA Momentum (Lendület) Program. This is his tenth paper appearing in IEEE/ACM Transactions on Networking.