Accurate Traffic Splitting on SDN Switches

Ori Rottenstreich, Yossi Kanizo, Haim Kaplan and Jennifer Rexford

***Abstract*—Traffic splitting is essential for load balancing over multiple servers, middleboxes, and paths. Often the target traffic distribution is not uniform (e.g., due to heterogeneous servers or path capacities). A natural approach is to implement traffic split in existing rule matching tables in commodity switches. In this paper we conduct an analytical study to understand this ability of switches. To do that, we indicate on a surprising strong connection between the description of distributions in switches to signed representations of positive integers. We introduce an optimal algorithm that minimizes the number of rules needed to represent a weighted traffic distribution. Since switches often have limited rule-table space, the target distribution cannot always be exactly achieved. Accordingly, we also develop a solution that, given a restricted number of rules, finds a distribution that can be implemented within the limited space. To select among different solutions, we describe metrics for quantifying the accuracy of an approximation. We demonstrate the efficiency of the solutions through extensive experiments.**

***Index Terms*—Load Balancing; Software Defined Networking; Ternary Content Addressable Memory**

I. Introduction

Traffic splitting is a commonly required capability in modern networks for balancing traffic over multiple network paths or servers. Traditionally, load balancers rely on dedicated middleboxes, servers or hardware switches for traffic splitting [2], [3], [4], [5]. Equal-cost multi-path routing (ECMP) [6], [7] is a common approach to achieve a uniform distribution by hashing the packet header. While ECMP achieves a uniform distribution, sometimes the desired distribution is not uniform. When servers are heterogeneous, more traffic should be sent to servers with more resources (e.g., CPU, memory, and storage). In irregular topologies, the network may need to split traffic unevenly among output ports when different paths have different costs. Furthermore, even regular topologies (e.g., fat-trees) can become irregular upon a link or switch failure.

WCMP (weighted cost multipathing) [8] generalizes ECMP for non-uniform distributions. While also relying on hashing, a variable number of entries is required for implementing the different distributions. Achieving high accuracy for skewed distributions, sometimes requires an unrealistic number of memory entries (e.g., at least proportional to the ratio between the largest and smallest weights).

One can also consider addressing load balancing through maintaining some state within stateful switching devices such as those supporting P4. A recent study [9] indicates that maintaining a state for all active connections, is typically impossible in the memory available in latest generations of switching ASICs. Authors mentioned that the capacity of forwarding tables is typically larger than that of stateful memories available to implement stateful algorithms in switches. Moreover, our approach also allows the aggregation of multiple connections in a single memory entry. Another potential issue is that to maintain per-connection consistency in stateful memories during updates, some connection states might need to be maintained externally in an expensive process. On the contrary, consistent updates of the content of forwarding tables can be relatively simple [10].

In the last years several schemes capitalize on the rule matching tables (implemented typically by Ternary Content Addressable Memory (TCAMs)), commonly available in commodity switches, to implement traffic splitting (e.g., [11] and Niagara [12]). A part of the packet header (e.g., the destination or the source IP field) is compared in parallel against a list of rules and traffic is forwarded according to the highest-priority matching rule. (Priority is usually implemented by ordering the rules, early rules in the order are of higher priority.) We address the problem of how to construct such tables that implement exactly or approximately a given distribution.

We restrict the tables to consist of prefix rule (wildcards are consecutive at the end of the rule). While TCAMs support general wildcard matches, common approaches for policy representations are limited to the use of prefix rules (e.g., [11], [13]). A critical reason is that finding a concise representation of a given mapping has a polynomial time algorithm for prefix rules [14], [15] while the problem is known to be NP-hard for wildcard matching [16]. Moreover, following the high power consumption of TCAMs such restricted implementations might allow some power saving [17], [18]. This typical restriction also appears in Niagara [12]. Likewise, recently suggested programmable switch architectures such as RMT and Intel's FlexPipe [19], [20] include various match-action tables and in particular tables restricted to prefix matching.

Assume traffic has to be split into $k = 3$ servers in ratio of 2:3:5 based on matching $W = 8$ bits of the header. This implies a target (unnormalized) distribution of $C = (0.2 \cdot 2^W, 0.3 \cdot 2^W, 0.5 \cdot 2^W)$. Assume that the $W = 8$ traffic bits are uniformly distributed with values in $\{00000000, \ldots, 1111111\}$. As illustrated in Table I, with three allowed rules, our target distribution C is best approximated as $D_1 = (64, 64, 128) = (0.25 \cdot 2^W, 0.25 \cdot 2^W, 0.5 \cdot 2^W)$ meaning that 64 bit combinations are mapped to server 1, another 64 bit combinations are mapped to server 2, and the remaining 128 are sent to server 3. With four rules a distribution of $D_2 = (48, 80, 128) = (0.1875 \cdot 2^W, 0.3125 \cdot 2^W, 0.5 \cdot 2^W)$ can be implemented, with a higher *similarity* to C (as formally

Manuscript received May 1, 2018; revised August 19, 2018. A partial version of this paper has been presented at the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), Vienna, Austria, July 2018 [1].
Ori Rottenstreich is with the department of Computer Science and the department of Electrical Engineering, Technion (e-mail: or@cs.technion.ac.il). Yossi Kanizo is with the department of Computer Science, Tel-Hai College (ykanizo@telhai.ac.il). Haim Kaplan is with the School of Computer Science, Tel Aviv University (haimk@post.tau.ac.il). Jennifer Rexford is with the department of Computer Science, Princeton University (jrex@cs.princeton.edu).

Target distribution C $= (0.2 \cdot 2^W, 0.3 \cdot 2^W, 0.5 \cdot 2^W) \approx (51, 77, 128)$		
$n_1 = 3$ rules	$n_2 = 4$ rules	$n_3 = 6$ rules
00****** $\rightarrow 1$	0000**** $\rightarrow 2$	00000000 $\rightarrow 1$
01****** $\rightarrow 2$	00****** $\rightarrow 1$	0000001* $\rightarrow 1$
1******* $\rightarrow 3$	01****** $\rightarrow 2$	0000**** $\rightarrow 2$
	1******* $\rightarrow 3$	00****** $\rightarrow 1$
		01****** $\rightarrow 2$
		1******* $\rightarrow 3$
Output distribution D		
$D_1 = (64, 64, 128)$	$D_2 = (48, 80, 128)$	$D_3 = (51, 77, 128)$

TABLE I

APPROXIMATING THE TARGET DISTRIBUTION C WITH A LIMITED NUMBER OF n RULES. RULES ARE DEFINED ON $W = 8$ BITS AND ARE ORDERED ACCORDING TO THEIR PRIORITIES. THE FIRST MATCHING RULE APPLIES.

defined later). With six rules an even closer distribution $D_3 = (51, 77, 128) \approx (0.1992 \cdot 2^W, 0.3008 \cdot 2^W, 0.5 \cdot 2^W)$ to C can be implemented.

If traffic is split based on the destination IP for instance, $W = 32$ in IPv4 and $W = 128$ in IPv6. Accordingly, exactly implementing a distribution by dedicating a rule for each of the 2^W possible bit combinations is impractical. TCAM memories are often restricted to thousands of rules (for instance, the above mentioned Intel's FlexPipe architecture [20] includes tables with up to 64K prefix rules), such that the majority of the memory is used for other tasks like classification. A critical reason that is high power consumption of TCAMs, known to be roughly proportional to their number of entries [21]. While an application might require a representation of high accuracy, little is known about the number of rules required to perform traffic split to within some prespecified accuracy, and how to optimally utilize a given number of rules to maximize accuracy. There are two previous papers, which we are aware that address the problem of how to split traffic by rule matching. The work of [11], uses disjoint rules, that is each packet is matched only by a single rule. This unnecessary restriction increases the table size. The work of Kang et al. [12] suggests an algorithm named Niagara. They did not provide any theoretical guarantees for Niagara but demonstrated empirically that it efficiently generates compact table. We conjecture that Niagara (or a slight variation of it) does compute the smallest set of rules required to exactly implement a given distribution (in which the probabilities are multiples of $1/2^W$).

Our contributions. In Section II we formalize the following two basic optimization problems: 1) The *Exact* problem: Find the smallest set of prefix rules that implement a given target distribution (assuming it is implementable). 2) The *Approximate* problem: Given a target distribution and a restricted number of rules, find the set of rules implementing a distribution which is "closest" to the target distribution among all distributions implementable within the constrained rule number. We consider two metrics to measure the distance between distributions. A first metric refers to the server with a maximal excess traffic while the second metric refers to the average error in the amount of allocated traffic over servers. The difficulty of the problems highly depends on the number of servers the distribution is defined for and accordingly we take this number into account in the design of our approach.

We first consider the case of splitting traffic to two servers in Section III. In this case we give efficient algorithms computing optimal solutions for the *Exact* and *Approximate* problems (for both metrics). We show a connection between the optimal solution for the *Exact* problem to particular signed bit representations of integers [22], [23]. Specifically, we characterize the number of rules in the optimal solution of the *Exact* problem in terms of the smallest weights of a signed bit representations of the integers specifying the distribution. This characterization also suggests how to obtain an optimal set of rules. One can verify that the solution computed by Niagara for the *Exact* problem given two servers obeys our characterization and is therefore optimal. For the *Approximate* problem we observe that for two servers our two metrics are the same, and we use the relation with the signed representations to give an optimal algorithm also for this problem.

We generalize our approach to the case of an arbitrary number of servers in Section IV, V and describe an optimal algorithm for the *Exact* problem that is efficient for a small number of servers. To do that, we first introduce a representation of a distribution over multiple servers by describing rule interactions between all pairs of servers. We show that the optimal solution can be found while restricting these interactions to be of a specific form. In all our experiments we observed that the number of rules that our algorithm computes is identical to the number of rules computed by Niagara, which supports the conjecture we made above. Finding an optimal polynomial time algorithm for the *Approximate* problem for an arbitrary number of servers (or proving that it is NP-hard) is an intriguing open question.

II. TRAFFIC SPLITTING PROBLEM

The input is a target traffic distribution $C = (c^1, \ldots, c^k)$ describing the relative amount of traffic required by each of the k servers $[1, k]$. Traffic is split between servers based on matching rules examining a field in the packet header. Let W describe the length in bits of this field in the header. We assume that $c^i > 0$ ($\forall i \in [1, k]$) and $\sum_i c^i = 2^W$, so the target distribution $C = (c^1, \ldots, c^k)$ is already "rounded" and specified by integers c^i's that sum up to 2^W, which is the total number of bit combinations in the designed header field.

The field value is assumed to be uniformly distributed. That is every bit combination among the 2^W possible ones has the same probability to appear in traffic. While this assumption is not generally true, we observe that: (i) There are some bits for which this is true, e.g., the least significant bits in many cases. That's enough for us and we can simply refer only to those bits; (ii) in some applications such bits can be achievable through the use of hash function (supported in recent versions of the P4 switch programming language [24]); Or (iii) there are ways to generalize some of our schemes to work around known non-uniformities which we will address in future work.

A matching rule is of the form $(s_1 \ldots s_W) \rightarrow a$ where $s_i \in \{0, 1, *\}$ and the wildcard $*$ stands for a don't care. It is composed of a matching pattern $(s_1 \ldots s_W)$ and an index $a \in [1, k]$ of a server. Rules are assumed to be of the form of *prefix rules* where we refer to the matching pattern simply as a prefix. A prefix rule has a prefix of length $\ell \in [0, W]$ describing the number of first bits it examines and $s_i = *$ for $i \in [\ell+1, W]$. We say that a packet with a bit combination $b_1 \ldots b_W$ (as its field value) matches a rule $s_1 \ldots s_W$ iff $s_i = b_i$ $\forall i \in [1, \ell]$. Intuitively, a prefix rule of length $\ell \in [0, W]$ corresponds to a subtree of size $2^{W-\ell}$ in the W-bits trie. The subtree includes the bit combinations the rule matches. The rule matching process relies on a semantics named *Longest Prefix Match (LPM)*. In case of a match in multiple rules, the longer more specific one is prioritized. Rules are ordered in a non-increasing order of their prefix lengths so that the first among multiple matches is with the longest prefix.

We refer to the number of bit patterns which are first matched by a rule as the *effective weight* of this rule. The effective weight determines the amount of traffic sent to the corresponding server based on the rule. We assume that all traffic is matched by at least one rule. We can see the set of rules as defining a function that maps each of the 2^W bit combinations $[0, 2^W - 1]$ in the header space to one of the k possible server indices. We refer to the distribution implied by the selected rules as $D = (d^1, \ldots, d^k)$ where $d^i \geq 0$ is the total amount of traffic to server i, i.e., the sum of the effective weights of rules pointing to server $i \in [1, k]$. We say that D is the *output distribution*. Since all traffic is assumed to be matched by at least one rule, D satisfies that $\sum_i d^i = 2^W$. Ideally, we would like to have $D = C$, meaning that $d^i = c^i$, $\forall i \in [1, k]$. Therefore our first optimization problem is defined as follows.

Problem 1. *Given a target distribution C for k servers. Find an* exact *representation of the function within a minimal number of rules.*

In many scenarios, the number of available rules is limited and it may be impossible to realize a specific target distribution C with the number of available rules. We define two metrics to measure the dissimilarity of C and D when $D \neq C$. The first is the maximum deviation of a server i *above* its target load c^i. The second is the average deviation or the ℓ_1 norm of $D - C$.

Definition 1. *(Dissimilarity Metrics) Consider a target distribution C for k servers. For a given output distribution D, a metric G examines the maximal amount of excess traffic in a server,*

$$G(D) = \max_{i \in [1,k]} \Big(\max\big(d^i - c^i, 0\big) \Big) = \max_{i \in [1,k]} \big(d^i - c^i\big).$$

Likewise, a metric H examines the average amount of error in the required traffic amount,

$$H(D) = \frac{1}{k} \cdot \sum_{i=1}^{k} |d^i - c^i|.$$

In the second optimization problem we are interested in a distribution with a constrained number of rules.

Problem 2. *Given a target distribution C for k servers and an upper bound n on the number of rules. Find a distribution D represented by at most n rules that minimizes $G(D)$ or $H(D)$.*

In particular, when we can implement C with at most n rules then $D = C$ and we have $G(D) = H(D) = 0$. Larger rule number n can enable finding a distribution closer to the target distribution, achieving smaller dissimilarities, for both metrics. For a given target distribution C and a number of allowed rules n we denote by G_{OPT}, H_{OPT} the optimal values of the metrics $G(D)$ and $H(D)$, respectively.

The same output distribution can be achieved by implementing different functions. For instance, the distribution $(\frac{1}{4} \cdot 2^W, \frac{3}{4} \cdot 2^W)$ can be obtained by the rules ($00{*}\ldots{*}{*}{*} \rightarrow$ server 1, $*{*}{*}\ldots{*}{*}{*} \rightarrow$ server 2) as well as by the rules ($11{*}\ldots{*}{*}{*} \rightarrow$ server 1, $*{*}{*}\ldots{*}{*}{*} \rightarrow$ server 2), describing two different mappings. Accordingly, *the requirement for a specific traffic distribution does not imply a unique mapping*.

This flexibility leads to an inherent difficulty. While it is easy to find a representation with a minimal number of (prefix) rules for a *particular mapping* (e.g., by the ORTC algorithm or similar alternatives [14], [15]), finding the most concise (exact) representation of a *target distribution* can be challenging since many possible mappings have to be considered. Furthermore, finding a closest representation given a specific number of rules can be even harder.

Since our model requires that all traffic is matched by the set of rules, we assume without loss of generality that the last among the n rules is a match-all (default) rule with a matching pattern $*{*}\ldots{*}{*}$. In any set of rules we can replace the last rule to be a match-all (without modifying its target) and get an equivalent set of rules of the same size.

Finally, throughout this paper, when looking for optimal rule sets, we can consider only sets given in a *compressed form* as defined below.

Definition 2. *A compressed-form prefix rule set is an ordered set of rules, where for each rule r_i in the set, the first colliding lower-priority rule r_j (i) has a shorter prefix, i.e. a larger number of wildcards, and, (ii) is mapped to a different server.*

By the above property of two intersecting prefixes, if condition *(i)* in Definition 2 is not met then r_j can be removed from the set, while if condition *(ii)* is not met then r_i can be removed; both remove operations do not affect the mapping that the original rule set implements. Note that a compressed form is not necessarily the most concise way to represent a function.

III. The case of two servers

In this section we consider the case of two servers. We present a simple mapping that realizes a given target distribution with a minimal number of prefix rules. This enables us to calculate the number of required rules for realizing a target distribution. Such information can help a network designer to estimate the number of rules available in a switch for other common tasks such as forwarding and traffic measurements. Furthermore, given a specific number of rules, we describe how to select them so that the distribution D which they realize minimizes $G(D)$ and $H(D)$. Let OPT_C be the minimal number of prefix rules required to obtain an output distribution that equals a target distribution $C = (c^1, c^2)$.

A. Representation as a range function

Different functions can be implemented to realize C. The next theorem shows that an optimal number of rules can always be achieved by a function that partitions the address space $[0, 2^W - 1]$ to two consecutive ranges, such that bit combinations from the first range are mapped to server 1 and bit combinations from the second range are mapped to server 2.

Theorem 1. *For a given target distribution $C = (c^1, c^2)$ with $k = 2$ servers, there exists a set of OPT_C rules implementing a function F_C satisfying $F_C(x) = 1$ for $x \in \left[0, c^1 - 1\right]$ and $F_C(x) = 2$ for $x \in \left[c^1, 2^W - 1\right]$.*

Proof. Consider an ordered set S of prefix rules that realizes the distribution $C = (c^1, c^2)$. We show how to construct an ordered set of prefix rules R implementing the function F_C such that *(i)* F_C satisfies $F_C(x) = 1$ for $x \in \left[0, c^1 - 1\right]$ and $F_C(x) = 2$ for $x \in \left[c^1, 2^W - 1\right]$, and *(ii)*, $|R| \leq |S|$.

Recall our Definition 2 regarding the compressed-form requirement. Furthermore, without loss of generality we assume that the last match-all rule in S is mapped to server 2. Thus, we can express the number of distinct bit combinations c^1 that are mapped to server 1 by using a linear combination of powers of two. Let b_i be the number of wildcards in the i-th rule, and define a_i as follows:

$$a_i = \begin{cases} 1 & i\text{-th rule is mapped to server 1,} \\ -1 & i\text{-th rule is mapped to server 2.} \end{cases}$$

Then, we get that

$$c^1 = \sum_{j=0}^{W-1} q_j \cdot 2^j, \qquad (1)$$

where $q_j = \sum_{i, b_i = j} a_i$, that is, q_j equals the total number of prefix rules with j wildcards that are mapped to server 1, minus those with j wildcards that are mapped to server 2.

We denote by $Q = (q_{W-1}, \ldots, q_1, q_0)$ the vector of the coefficients of the powers of two in Equation (1). Note that the shortest prefix, that is, the one with the largest number of wildcards, excluding the last match-all rule, may have at most $W - 1$ wildcards.

The elements in Q will be used to construct the alternative ordered set of prefix rules R that satisfies the required properties of this theorem. In this construction, in addition to a match-all rule, the number of rules in R will be exactly as the sum of the absolute values of the elements in Q. By the definition of Q, the number of rules in R will be at most the number of the rules in S.

We further simplify Q by operations on its elements:

- For $j \in [0, W - 2]$ such that $q_j \geq 2$ set $q_j := q_j - 2$ and $q_{j+1} := q_{j+1} + 1$.
- For $j \in [0, W - 2]$ such that $q_j \leq -2$ set $q_j := q_j + 2$ and $q_{j+1} := q_{j+1} - 1$.

Each of these two operations preserves Equation (1), and it lowers the sum of absolute values of the elements in Q by at least 1. We perform either of these operations repetitively until they do not apply anymore. The resulting vector Q satisfies that

- For the largest j for which $q_j \neq 0$, $q_j = 1$.
- The vector Q has at most W elements.
- For each $j \in 0, \ldots, W - 1$, $|q_j| \leq 1$.

Finally, we construct the prefix rule set R. We first select the last match-all rule as in S (mapped to server 2), and then we add rules with an increasing order of their priority by going over the elements of Q.

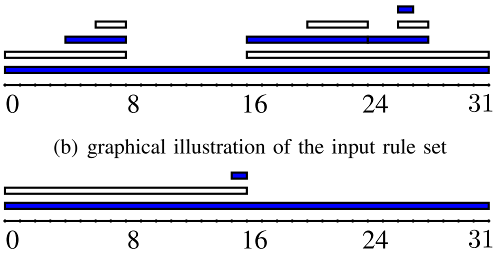
Specifically, set $u = 0$, and for each of the elements of Q (from $j = W - 1$ to $j = 0$):

- If $q_j = 1$, add the rule $\left[u, u + 2^j - 1\right] \rightarrow 1$, and set $u := u + 2^j$.
- If $q_j = -1$, add the rule $\left[u - 2^j, u - 1\right] \rightarrow 2$, and set $u := u - 2^j$.
- Skip if $q_j = 0$.

In the last construction, the number of rules in R is not larger than that of S, and the function F_C defined by the rule set R satisfies $F_C(x) = 1$ for $x \in \left[0, c^1 - 1\right]$ and $F_C(x) = 2$ for $x \in \left[c^1, 2^W - 1\right]$. Therefore, the theorem follows. □

#	rule	mapping	#	rule	mapping
1	11010	server 2	6	110**	server 2
2	0011*	server 1	7	10***	server 2
3	1101*	server 1	8	00***	server 1
4	001**	server 2	9	1****	server 1
5	101**	server 1	10	*****	server 2

(a) Input rule set (with 10 rules, defined over $W = 5$ bits), that yields a distribution of $(15, 17)$ of the $2^W = 32$ bit combinations.



(b) graphical illustration of the input rule set

(c) graphical illustration of the resulting rule set (with 3 rules) that describes a simpler function and yields the same distribution (15,17)

Fig. 1. An example of the technique described in the proof of Theorem 1: (a) an example rule set given in a compressed form defined over $W = 5$ bits, (b) a graphical illustration of the rule set, and (c) a graphical illustration of the resulting rule set (01111 → server 2, 0**** → server 1, ***** → server 2). Clear white and solid blue rectangles correspond to rules mapped to server 1 and server 2, respectively.

The construction of the new rule set R given the rule set S as described in the proof of Theorem 1 is illustrated in the following example.

Example 1. *Given the rule set S in Fig. 1(a) which is illustrated graphically in Fig. 1(b), the vector $Q = (q_4, q_3, q_2, q_1, q_0)$ is initially equal to $(1, 0, -1, 2, -1)$. The reason $q_2 = -1$ is due to the rules with 2 wildcards which are rules 4, 5, and 6. These rules are mapped to servers 2,1, and 2, respectively. Therefore, $a_4 = -1$, $a_5 = 1$, and $a_6 = -1$, which sum up to $q_2 = -1$. The number of bit combinations that are mapped to server 1 is given by $c^1 = 1 \cdot 2^4 + 0 \cdot 2^3 - 1 \cdot 2^2 + 2 \cdot 2^1 - 1 \cdot 2^0 = 15$.*

The only element with an absolute value greater or equal to 2 is q_1. Therefore, we apply the simplification process on q_1 and get that $q_1 := q_1 - 2 = 0$ and $q_2 := q_2 + 1 = 0$. The resulting vector $Q = (1, 0, 0, 0, -1)$ has no element with an absolute value greater than 1. Hence, this process is over.

*To construct the rule set R, we first take the match-all rule as in S (mapping to server 2). We set $u = 0$, and go from left to right over the elements of Q. For q_4 we add the rule $\left[0, 2^4 - 1\right] \rightarrow 1$, and set $u = 16$. We skip q_3, q_2, and q_1 since they equal 0, and last, for q_0, we add the rule $\left[2^4 - 2^0, 2^4 - 1\right] \rightarrow 2$. Fig. 1(c) shows a graphical illustration of the resulting rule set $S = (01111 \rightarrow$ server 2, 0**** $\rightarrow$ server 1, ***** $\rightarrow$ server 2$)$.*

B. Calculating the cost of a given distribution with signed representations of positive integers

Following Theorem 1, we express the minimal number of prefix rules OPT_C required to follow (exactly) a target distribution $C = (c^1, c^2)$ by relating it to signed representations of positive integers. As mentioned, this can be useful for a network designer to determine the number of rules available for other tasks such as forwarding and traffic measurements.

Unlike the regular binary representation, in the signed-bit representation an integer is described as a sum of positive and negative powers of two. We now define it formally, following the terminology of [23].

Definition 3. *A signed-bit representation of $y \in \mathbb{Z}$ is given by a sequence $Q = (q_t, q_{t-1}, \ldots, q_0)$, such that $y = \sum_{i=0}^{t} q_i \cdot 2^i$ and $\forall i \in [0, t-1], q_i \in \{-1, 0, 1\}$ and $q_t \in \{-1, 1\}$. We refer to $t + 1$ as the length of the representation and to the number of non-zero q_i's as the* weight *of the representation. The integer 0 is represented by the empty sequence denoted by ().*

Unlike the regular binary representation, which is unique, there are multiple signed-bit representations for a given integer $y \in \mathbb{Z}$. Consider for instance the integer $y = 7$. While the unique binary representation $(1, 1, 1)$ is also a signed-bit representation (satisfying $7 = 4 + 2 + 1 = 2^2 + 2^1 + 2^0$), another signed-bit representation is $(1, 0, 0, -1)$ (satisfying $7 = 8 - 1 = 2^3 - 2^0$). The last representation has a property captured in the following definition.

Definition 4. *A signed-bit representation of an integer $y \in \mathbb{Z}$ is said to be in a* non-adjacent form *if there are no two non-zero adjacent signed bits, that is, $\forall i \in [1, t]$, if $q_i \neq 0$ then $q_{i-1} = 0$.*

By [23] positive integers have a unique non-adjacent form. This can be easily generalized for any integer.[1]

Property 1. *All integers have a unique non-adjacent form representation.*

It is easy to derive the non-adjacent signed-bit form representation of an integer. Start with its binary representation and while beginning from the right bit, replace any sequence of $0, 1, 1, \ldots, 1, 1$ by the sequence $1, 0, 0, \ldots, 0, -1$ of the same length (where the most significant 1 bit of the assigned sequence can be considered as the least significant 1 bit of the next sequence to be replaced).

As we show later, we are interested in the weight of the representation since it relates to the number of prefix rules required to follow a distribution. The following property is due to [23].

Property 2. *For all integers, the non-adjacent form has a minimal weight among all signed-bit representations.*

Notice that for some integers, in addition to the unique non-adjacent form, there can be additional signed-bit representations that also achieve the minimal weight.

For an integer x, we denote by $\phi(x)$ the weight of its non-adjacent form representation. It is easy to calculate $\phi(x)$ by the above computation of the non-adjacent form. Clearly, $\phi(x) = \phi(-x)$. Notice that for a distribution $C = \left(c^1, c^2\right)$ of $k = 2$ servers, we have $c^1 + c^2 = 2^W$ and accordingly $|\phi(c^1) - \phi(c^2)| \leq 1$. To represent c^j in a signed bit representation we can always negate a representation of the other value c^{3-j} and add a coefficient of 2^W (achieving a representation with a weight that is at most larger by one and thus by Property 2 the weight of a non-adjacent form cannot be larger). We characterize the minimal required number of rules to exactly represent a distribution.

Theorem 2. *Consider a target distribution $C = \left(c^1, c^2\right)$. The minimal number of rules OPT_C required to realize C, is given by $\min(\phi(c^1), \phi(c^2)) + 1$.*

Proof. We first show that we can realize C with $\min(\phi(c^1), \phi(c^2)) + 1$ rules. Without loss of generality assume that the minimum is attained by $\phi(c^1)$. We use $\phi(c^1)$ rules (mapping prefixes either to server 1 or to server 2) to map c^1 bit combinations to server 1. If the last rule is not a match-all rule, we add a last match-all rule to map the other $c^2 = 2^W - c^1$ bit combinations to server 2. Thus $OPT_C \leq \min(\phi(c^1), \phi(c^2)) + 1$. For the opposite inequality, assume C is realized with OPT_C rules. Let's assume, without loss of generality, that the last is an all-match rule that maps to server 2. The value c^1 must correspond to the number of bit combinations matched by the rules that map to server 1 among the first $OPT_C - 1$ rules. We can derive from these rules a signed bit representation with a weight $OPT_C - 1$ for c^1. Thus $\phi(c^1) \leq OPT_C - 1$. □

Interestingly, the values of $\phi(x)$ (for non-negative values) have been widely investigated, and the values $\{\phi(x) \mid x \geq 0\}$ have been described as a sequence in the Encyclopedia of Integer Sequences [22]. Applications of the sequence have been suggested for instance for minimizing communication between processors as well as for routing in peer-to-peer networks [25], [26]. Bounds and recursive formulas for the value of $\phi(x)$ were suggested. An illustration of the values of $\phi(x)$ and OPT_C of a distribution $C = (x, 2^7 - x)$ for $x \in [0, 128]$ is given in Fig. 2. This graph can give an intuition on the representation cost of a distribution. It is easy

[1] Clearly, this property of positive integers applies for any integer since we can negate a represented number by negating the signed bits in its signed-bit representation. Similarly, there is only one representation for 0.

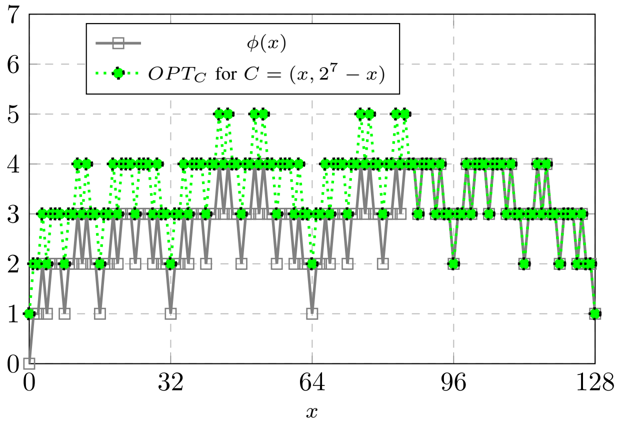


Fig. 2. The value of $\phi(x)$ and the number of rules OPT_C required to describe a distribution $C = (x, 2^7 - x)$ for an integer $x \in [0, 2^7]$.

to observe the symmetry of OPT_C, i.e., the same rule count is required for $(x, 2^7 - x)$ and $(2^7 - x, x)$. We can also see that $OPT_C \in [\phi(x), \phi(x) + 1]$ and that the only distributions that can be described by at most two rules are those where x or $2^7 - x$ are powers of two.

C. Approximated distribution realization

Consider a target distribution $C = (c^1, c^2)$ and a given number of allowed rules n. We study the case where the target distribution cannot necessarily be realized accurately by at most n rules (i.e., $n < OPT_C$). Instead, we find a realizable distribution with a minimal dissimilarity value with the target. By Theorem 2, the output distribution $D = (d^1, d^2)$ must satisfy $\min(\phi(d^1), \phi(d^2)) + 1 \leq n$. For the metric G, considering the maximal amount of excess traffic in a server, among the realizable distributions (d^1, d^2) we would like to find the one minimizing $|d^1 - c^1| = |d^2 - c^2|$. For the metric H, considering the average amount of error, we would like to minimize the sum $0.5 \cdot (|d^1 - c^1| + |d^2 - c^2|) = |d^1 - c^1|$. It follows that for $k = 2$ servers the two metrics are minimized by the same distributions. In the rest of this section we describe an efficient algorithm that achieves a target distribution minimizing the two metrics.

We start with a statement on the number of bits required to represent an integer in its non-adjacent form. Intuitively, it shows that given $t + 1$ bits, the largest integer that can be represented (in its non-adjacent form) is $y_u = 2^t + 2^{t-2} + \ldots$, that is starting with 1 in the most significant bit and alternating between 1 and 0 when going from left to right. Likewise, the smallest integer that can be represented is $y_d = -y_u = -2^t - 2^{t-2} - \ldots$. Furthermore, the next lemma shows that the non-adjacent form of any integer in the range $[y_d, y_u]$ has no more than $t + 1$ bits.

Lemma 3. *An integer y has a non-adjacent form representation of at most $t + 1$ bits iff*

(i) $|y| \leq 2^t + 2^{t-2} + \ldots + 1$ *for an even t,*
(ii) $|y| \leq 2^t + 2^{t-2} + \ldots + 2$ *for an odd t.*

Proof. The proof is by an induction on $t \geq 0$. For $t = 0$, the only possible non-adjacent forms of length at most 1 are (1), $()$ and (-1), representing the integers 1, 0 and -1, respectively. For $t = 1$, there are two additional non-adjacent representations of length 2, namely $(1, 0)$ and $(-1, 0)$, representing 2 and -2, respectively. For the induction step, assume that the claim holds for $t - 1$ and $t - 2$, the proof for t is as follows. We assume that t is even, the proof for odd t is the same.

Let y be the largest number whose non-adjacent representation consists of $t+1$ bits. The representation of y has bit t equal to 1, bit $t - 1$ equal to 0, and the rest $t - 1$ bits representing the largest integer, y', whose non-adjacent representation is of length $t - 1$. It follows that $y = 2^t + y'$ which by induction equals to $2^t + 2^{t-2} + \ldots + 1$. The proof that the smallest number that can be represented with $t+1$ bits is $-(2^t + 2^{t-2} + \ldots + 1)$ is analogous.

For the converse, let y be a number such that $|y| \leq 2^t + 2^{t-2} + \ldots + 1$. We show a non-adjacent representation of y with at most $t + 1$ bits. If $y \geq 2^t - (2^{t-2} + \ldots + 1)$ then we set bit t to 1, bit $t - 1$ to 0, and the rest $t - 1$ are set such that they represent $y' = y - 2^t$. Since $|y'| \leq 2^{t-2} + \ldots + 1$ such a representation for y' exists by the induction hypothesis. The argument for $y \leq -2^t + (2^{t-2} + \ldots + 1)$ is analogous. If $y < 2^t - (2^{t-2} + \ldots + 1)$ and $y > -2^t + (2^{t-2} + \ldots + 1)$ then $y \leq 2^{t-1} + 2^{t-3} + \ldots + 2$ and $y \geq -(2^{t-1} + 2^{t-3} + \ldots + 2)$ so we get that y can be represented by t bits by induction. □

Recall that we aim at finding a set of at most n rules that best approximates the target distribution $\left(c^1, c^2\right)$. The following lemma significantly reduces the search space for the output distribution $D = \left(d^1, d^2\right)$.

Lemma 4. *Given an integer $y = x \cdot 2^a$, with $a, x \in \mathbb{N}$, and let $U_a = \sum_{i=1}^{\lfloor a/2 \rfloor} 2^{a-2\cdot i}$. Then,*

$$\min \{\phi(y - U_a), \ldots, \phi(y), \ldots, \phi(y + U_a)\} = \phi(y) = \phi(x).$$

Moreover, the value of $\phi(y)$ is uniquely retrieved for y.

Proof. Let X be the non-adjacent form of x. We first show that all integers in the range $[y - U_a, y + U_a]$ have the same prefix X followed by (at least one) bit of 0. Consider the non-adjacent form representation of y. It has a prefix X followed by exactly a zeros. By applying Lemma 3 with $t = a - 2$, all integers whose absolute value is smaller or equal to $U_a = \sum_{i=1}^{\lfloor a/2 \rfloor} 2^{a-2\cdot i}$ have a non-adjacent form representation using upto $a - 1$ bits. For each such integer z, since its non-adjacent form has upto $a - 1$ bits, the non-adjacent form of $y + z$ differs from the non-adjacent form of y only in the lower $a - 1$ bits. Therefore, all integers in the range $[y - U_a, y + U_a]$ have the same prefix X, followed by (at least one) bit of 0. Since all a least significant bits of the non-adjacent form of y are zeros, and it is the only integer with that property, it has the minimal weight, which is uniquely retrieved only for it, and also equals $\phi(x)$. □

To find a distribution $D = (d^1, d^2)$ satisfying $\min(\phi(d^1), \phi(d^2)) + 1 \leq n$, we consider four scenarios such that at least one of them occurs (the scenarios are not necessarily disjoint). We explain how to find D of a minimal dissimilarity under each scenario. They are *(i)* $d^1 \geq c^1$ and $\phi(d^1) = \min(\phi(d^1), \phi(d^2))$, *(ii)* $d^1 \leq$

c^1 and $\phi(d^1) = \min(\phi(d^1), \phi(d^2))$, *(iii)* $d^1 \geq c^1$ and $\phi(d^2) = \min(\phi(d^1), \phi(d^2))$, *(iv)* $d^1 \leq c^1$ and $\phi(d^2) = \min(\phi(d^1), \phi(d^2))$.

Since for $k = 2$ servers the two metrics G and H are minimized by the same distributions, we arbitrarily focus on the metric G and the optimality follows also for the metric H. We discuss scenario *(i)*. We consider $W + 1$ disjoint ranges for the value $d^1 \in [c^1, 2^W]$. The ranges are denoted by $R_0, R_1, \ldots, R_W$ such that $R_a = [\lceil c^1/2^a \rceil \cdot 2^a - U_a, \lceil c^1/2^a \rceil \cdot 2^a + U_a]$ for $a \in [0, W]$. Let ϕ_a be the minimal value of ϕ for values in R_a. By Lemma 4 it satisfies $\phi_a = \phi(\lceil c^1/2^a \rceil \cdot 2^a)$. We take the minimal value of a that satisfies $\phi_a \leq n - 1$. Notice that for $a \in [0, W - 1]$ it satisfies $\phi_a - \phi_{a+1} \leq 1$. This is since the two values $\lceil c^1/2^a \rceil \cdot 2^a$, $\lceil c^1/2^{a+1} \rceil \cdot 2^{a+1}$ are either equal or differ by the power of two 2^a. For the selected value of a, we set d^1 as $\lceil c^1/2^a \rceil \cdot 2^a$ and we have that this value minimizes the error while satisfying the constraint of n. The scenario of *(ii)* is similar. We consider $W + 1$ disjoint ranges for the value $d^1 \in [0, c^1]$. They are $R_0, R_1, \ldots, R_W$ such that $R_a = [\lfloor c^1/2^a \rfloor \cdot 2^a - U_a, \lfloor c^1/2^a \rfloor \cdot 2^a + U_a]$. We find the first range for which $\phi_a = \phi(\lfloor c^1/2^a \rfloor \cdot 2^a) \leq n - 1$. Then we set $d^1 = \lfloor c^1/2^a \rfloor \cdot 2^a$. For *(iii), (iv)* we repeat $(i), (ii)$ by replacing c^1, c^2. Finally, among the four options, we select the one minimizing $|d^i - c^i|$.

IV. The vector-set representation for multiple servers

We study the case of an arbitrary number of servers. Our ultimate goal is to develop also for this scenario solutions for an exact representation with minimal rules or the best representation for a given number of rules. Towards this goal, while relying on an analytic model, we suggest a novel representation of a given rule set which can be manipulated to construct an alternative low-cost rule set that yields the same distribution. Then, in Section V we use this tool to develop algorithms for both problems.

In Section III, for the case of two servers, we used the vector $Q = (q_{W-1}, \ldots, q_1, q_0)$ with coefficients of powers of two for summarizing a set of rules involving two servers. In this section, we generalize this representation for multiple servers. We refer to this generalization as a *vector set*, denote it by $\hat{Q}$ and explain that a vector set implies a single distribution. In Section IV-A, we formally define the vector set, explain how to construct it for a given set of rules and study its properties. Then, in section IV-B, we explain how to process a vector set while keeping the distribution it implies, so that it can be realized into a set of rules of a small size.

A. Construction and basic properties

A vector set $\hat{Q}$ consists of k^2 vectors denoted as $\{Q^{ij}\}$ with $i, j \in [1, k]$. Each vector $Q^{ij} = \left(q^{ij}_{W-1}, \ldots, q^{ij}_0\right)$ has W elements. A given set of rules, can be associated with a vector set $\hat{Q}$, described in the following. Informally, a vector Q^{ij} represents the amount of traffic (number of bit combinations) that server i "takes" from server j. Thus, a vector set $\hat{Q}$ represents the entire relation (in that manner) between the servers.

#	rule	mapping
1	11011	server 2
2	0011*	server 1
3	1101*	server 3
4	001**	server 2
5	110**	server 1
6	10***	server 2
7	01***	server 1
8	*****	server 3

$$\begin{aligned} Q^{12} &= (0,0,0,1,0) \\ Q^{13} &= (0,1,1,-1,0) \\ Q^{23} &= (0,1,1,0,1) \end{aligned}$$

Fig. 3. Rule set example (left) and its corresponding vector set representation (right). The output distribution is $D = (12, 11, 9)$.

We explain a way to construct a vector set from a general compressed-form rule set S for representing its structure. Formally, consider a general ordered set S of prefix rules. We assume that the rule set S adheres to the compressed-form requirement described in Definition 2 and that the last match-all rule is mapped to server k. Following Definition 2, for each rule r in the set, the first colliding lower-priority rule, that is, the rule that r "takes" traffic from, *(i)* has more wildcards, and *(ii)* maps to a different server.

The construction of the vectors in $\hat{Q}$, given a set of rules is defined by the following process: Initiate all vectors in $\hat{Q}$ to zero, and repeat the following for each rule starting from the highest priority rule (excluding the match-all rule). For each rule, denote by i the server it maps to and by z its number of wildcards. Find its first lower-priority colliding rule and denote its server by j. Then, increase $q^{i,j}_z$ and decrease $q^{j,i}_z$, both by one. These operations reflect the fact that server i eliminates 2^z bit combinations from server j.

By the definition of the above construction, for all $i, j \in [1, k]$ and $z \in [0, W - 1]$, $q^{i,j}_z = -q^{j,i}_z$. Moreover, since for each rule (excluding the match-all rule), its first lower-priority colliding rule is mapped to a different server then for all $i \in [1, k]$ and $z \in [0, W - 1]$, $q^{i,i}_z = 0$.

Fig. 3 shows an example of a rule set S and its corresponding vector set representation $\hat{Q}$. Since Q^{21}, Q^{31} and Q^{32} are the element-wise negation of Q^{12}, Q^{13} and Q^{23}, only the latter vectors are shown; the vectors Q^{11}, Q^{22}, and Q^{33} are all zeroed.

The construction of the vectors (initialized with zeros) starts with the first rule 11011 that has 0 wildcards and maps to server 2. Its first lower-priority colliding rule is rule 3 (1101$*$, mapped to server 3). Therefore, we increase by one $q^{2,3}_0$ (and decrease $q^{3,2}_0$). Next, the first colliding rule of rule 2 (mapping to server 1, with a single wildcard) is rule 4 (mapping to server 2), then we increase $q^{1,2}_1$ (and decrease $q^{2,1}_1$). This process continues for all rules (excluding the match-all rule).

Intuitively, if one keeps track on the exact function implemented by considering only the last t rules for $t = 1, 2, ..., k$, the vector set $\hat{Q}$ represents succinctly, using cancellation, the number of times there is a change in the function implemented by the rule set. In particular, a vector Q^{ij} represents the times the change in function involves server i and j.

Accordingly, one can count the number of bit combinations mapped to each server. Let $T^{ij} = \sum_{t=0}^{W-1} q^{ij}_t \cdot 2^t$. The value T^{ij} counts the total number of bit combinations server i takes

$$\begin{array}{rcl} Q^{ix} = (\_,\_,\geq 1,\_,\_) & & \Delta Q^{ix} = (\_,\_,-1,\_,\_) \\ Q^{iy} = (\_,\_,\_,\_,\_) & \Rightarrow & \Delta Q^{iy} = (\_,\_,+1,\_,\_) \\ Q^{xy} = (\_,\_,\geq 1,\_,\_) & & \Delta Q^{xy} = (\_,\_,-1,\_,\_) \end{array}$$

(a) step I

$$Q^{ix} = (\_,\_,\geq 2,\_,\_) \quad \Rightarrow \quad \Delta Q^{ix} = (\_,+1,-2,\_,\_)$$

(b) step II

$$\begin{array}{rcl} Q^{ix} = (\_,\_,\geq 1,\_,\_) & & \Delta Q^{ix} = (\_,+1,-1,\_,\_) \\ Q^{iy} = (\_,\_,\geq 1,\_,\_) & \Rightarrow & \Delta Q^{iy} = (\_,\_,-1,\_,\_) \\ Q^{xy} = (\_,\_,\_,\_,\_) & & \Delta Q^{xy} = (\_,\_,+1,\_,\_) \end{array}$$

(c) step III

Fig. 4. Illustration of the simplification process of the vector set $\hat{Q}$, describing the delta (addition) to each of the vectors. The output distribution is preserved in each of these changes.

from server j. Given these values one can count for each server i the total number of bit combinations d^i that the function maps to:

$$d^i = \begin{cases} \sum_{j=1}^{k} T^{ij} & 1 \leq i \leq k-1 \\ 2^W + \sum_{j=1}^{k} T^{ij} & i = k \end{cases}$$

For each vector Q^{ij}, we further define a partial (weighted) sum series of its elements. For $v \in [0, W-1]$, representing prefix length, let $T_v^{ij} = \sum_{t=v}^{W-1} q_t^{ij} \cdot 2^t$ and $T_W^{ij} = 0$. An equivalent more intuitive definition is through using the following recursion: Let $T_{W-1}^{ij} = q_{W-1}^{ij} \cdot 2^{W-1}$, and for $v \in [0, W-2]$, $T_v^{ij} = T_{v+1}^{ij} + q_v^{ij} \cdot 2^v$. Likewise, let $q_t^i = \Sigma_{j \in [1,k]} q_t^{ij}$. Last, we define the number of bit combinations mapped to each server by rules with prefix length of at most $W-1-v$ (namely more than v wildcards), as represented by the vectors in $\hat{Q}$:

$$d_v^i = \begin{cases} \sum_{j=1}^{k} T_v^{ij} = \sum_{t=v}^{W-1} q_t^i \cdot 2^t & 1 \leq i \leq k-1 \\ 2^W + \sum_{j=1}^{k} T_v^{ij} = 2^W + \sum_{t=v}^{W-1} q_t^i \cdot 2^t & i = k \end{cases}$$

We capture a simple property of a vector set.

Theorem 5. *Given a compressed-form prefix rule set S, for all $i \in [1,k], v \in [0,W]$, $d_v^i \geq 0$.*

Proof. Since the rule set S adheres to the compressed-form requirement, we can assume that rules are ordered by a non-increasing prefix length. Consider the series of functions the rule set implements when going over the rules by an increasing order of their prefix length (from 0 to W), where at each step we add all rules with a certain prefix length. In considering the intermediate function of each step, the number of bit combinations mapped to every server must be at least 0, and the claim follows by the definition of d_v^i. □

B. Processing and realization

We describe a technique to reduce the number of rules required to achieve the output distribution of a vector set. In the next theorem we show that the vector set can be manipulated, preserving its original implemented distribution, such that for each prefix length and for each server there is at most one rule that changes the number of bit combinations mapped to the server.

Theorem 6. *Any vector set $\hat{Q}$, with $d_v^i \geq 0$ for all $i \in [1,k]$ and $v \in [0, W-1]$, can be processed, preserving the original distribution and the non-negativity of its partial sums such that $q_t^{ix} \in \{-1,0,1\}$ for all i,x,t. Further, for all t and i, if for some x, $q_t^{ix} \neq 0$, then for all $j \neq x$, $q_t^{ij} = 0$ (and $q_t^{ji} = 0$).*

Proof Outline. The processing has two main phases, each composed of several steps among steps I-III, as illustrated in Fig. 4. Each of the steps maintains the output distribution D. We verify that along the processing, for all $i \in [1,k], v \in [0,W]$ the partial sums satisfy $d_v^i \geq 0$. Phase 1 relies on steps I and II. In step I, for instance, illustrated in Fig. 4(a), we consider $t \in [0, W-1]$. Assume there exist i,x,y such that $q_t^{ix}, q_t^{xy} > 0$. We reduce q_t^{ix}, q_t^{xy} by one and increase q_t^{iy} by one (and update $q_t^{xi}, q_t^{yx}, q_t^{yi}$ correspondingly). In phase 1, we repeat steps I and II, column by column for $t \in [0, W-2]$ and then apply step I for $t = W-1$. In phase 2, steps I and III are repeated, column by column for $t \in [0, W-2]$ and then step I is applied for $t = W-1$. We explain that following phases 1 and 2, the vector set has the required form in all columns besides maybe the most-left one. To satisfy the property also for that column, we might have to replace the default server by another server. □

The next theorem shows that when a simple condition on a vector set holds, there exists a set of rules (in a compressed form) for which the vector set corresponds.

Theorem 7. *Consider a vector set $\hat{Q}$ satisfying: (i) $d_v^i \geq 0$ for all $i \in [1,k], v \in [0,W]$. (ii) $q_t^{ix} \in \{-1,0,1\}$ for all i,x,t. (iii) for all t and i, if for some x, $q_t^{ix} \neq 0$, then for all $j \neq x$, $q_t^{ij} = q_t^{ji} = 0$. Then, vector set can be realized to a compressed-form prefix rule set.*

Proof. Assume that we are given a vector set $\hat{Q}$ such that for all $i \in [1,k], v \in [0,W]$, $d_v^i \geq 0$. We show a construction of a compressed-form rule set. Note that the prefix length of a rule with v wildcards is $W-v$. The construction is in $W+1$ steps considering values of $v = W, W-1, ..., 0$. For an iteration with value v, rules of v wildcards are added (with higher priority) to those obtained in previous steps. We show that the set of rules in step v yields an output distribution of $(d_v^1, d_v^2, ..., d_v^k)$ and results in a vector set obtained from $\hat{Q}$ by setting to zero in all vectors the values with indices smaller than v. First for $v = W$, we start with a single match-all rule (with W wildcards) mapping traffic to server k. For $v < W$, start with the set of rules for the previous step for $v+1$. Consider the values q_v^{ij} for $i,j \in [1,k]$ for which $q_v^{ij} > 0$. Add q_v^{ij} prefix rules with v wildcards, mapping traffic to server i that contradicts a rule for server j. The added rules should have the first $W-v$ bits as the rule for server j. The rules have additional non-wildcard bits to have v wildcards, while the following non-wildcard bits are selected as the minimal values that avoid an intersection with rules previously added for also taking traffic from server j. By the positiveness of the values d_v^i we can find such rules for server i to contradict a rule associated with server j. Note that for each column at most one rule is required. The set of rules obtained in the last step of $v = 0$ (with 0 wildcards) is the desired one. □

#	rule	mapping
1	11000	server 2
2	1100*	server 3
3	100**	server 1
4	00***	server 1
5	1****	server 2
6	*****	server 3

$$\begin{aligned} Q^{12} &= (0,0,1,0,0) \\ Q^{13} &= (0,1,0,0,0) \\ Q^{23} &= (1,0,0,-1,1) \end{aligned}$$

Fig. 5. The rule set S and the corresponding vector set representation $\hat{Q}$ derived after the processing of the rule set from Fig. 3. The output distribution is again $D = (12, 11, 9)$.

B^i_{W-u-1}	q^i_{W-u-1}	server-state for bit $u+1$
0	-1	positive
0	0	invalid
0	1	invalid
1	-1	negative / zero
1	0	positive
1	1	invalid

TABLE II

SERVER STATE TRANSITION: DEPENDENCY OF SERVER STATE i FOR BIT $u+1$ ON $B^i_{W-(u+1)}$ AND q^i_{W-u-1}, GIVEN A POSITIVE-STATE FOR BIT INDEX $W-u$.

Note that the number of required (non-default) rules in the construction of the proof of Theorem 7 equals half the sum of the absolute values of all elements (i.e. $0.5 \cdot \sum_{i,j \in [1,k]} |q^{i,j}|$). Fig. 5 describes the set of rules obtained after the processing of the vector set from Fig. 3. While maintaining the output distribution, the number of rules is reduced from 8 to 6.

V. Solutions for Multiple Servers

Inspired by the representation of a distribution for multiple servers through a vector set from Section IV, we turn to design algorithms that find an exact representation with minimal rules (in Section V-A) and the best representation for a given number of rules (in Section V-B).

A. Exact distribution realization

We describe an algorithm to find an exact representation with the minimal possible number of rules for any given target distribution. We start with properties that relate the vector set to the output distribution it yields. For space constraints we provide the high-level details of the algorithm.

Theorem 8. *Let $\hat{Q}$ be a vector set with an output distribution $D = (d^1, \ldots, d^k)$ for which the processing from Theorem 6 was applied. For all $i \in [1,k], u \in [1,W]$, let $h^i_u = \lfloor d^i / 2^{W-u} \rfloor \cdot 2^{W-u}$. Then, the value d^i_{W-u}, expressed by the u high-indices values of the vector set, satisfies $d^i_{W-u} = h^i_u$ or $d^i_{W-u} = h^i_u + 2^{W-u}$.*

Proof. By definition $d^i_{W-u} = d^i - \sum_{t=0}^{W-u-1} 2^t \cdot q^i_t \in [d^i - (2^{W-u} - 1), d^i + (2^{W-u} - 1)]$, where the bounds follow the processing of the vector set $\hat{Q}$. Likewise, $h^i_u \in [d^i - (2^{W-u} - 1), d^i]$. Since d^i_{W-u} and h^i_u are both multiplies of 2^{W-u}, the result follows. □

We define the notion of a *server state*. Given a vector set $\hat{Q}$, a server $i \in [1,k]$ is associated with a state for every $u \in [0,W]$ (corresponds to the bit index $W-u$). Intuitively, the state examines the difference between the allocation of a server following the complete vector set $\hat{Q}$ and its allocation as expressed by some high-indexed bits of $\hat{Q}$. For $u \in [1,W]$,

- A server i is in *zero-state* for u if $d^i_{W-u} = h^i_u = d^i$.
- In *negative-state* for u if $d^i_{W-u} = h^i_u$ and $d^i_{W-u} < d^i$.
- In *positive-state* for u if $d^i_{W-u} = h^i_u + 2^{W-u}$. This implies that $d^i < d^i_{W-u}$.

In other words, the server i is in positive state for a bit index $W-u$ if all rules involving it with prefix-length lower than u encode a specific number that is larger than the number of bit combinations in the target distribution. It is in a negative state if these rules encode a specific number that is lower than the number of bit combinations in the target distribution, and it is in zero state if both numbers are equal. By Theorem 8, there is no other possibilities.

Assuming that none of the servers has a target number of bit combinations equals zero, then by this definition it follows that for $u = 0$ all servers are in negative-state, except for the server that is assigned with the default rule who is in positive state.

For each of the servers, its states (either negative, positive, or zero) over the bit indices are related to each other.

Given the state of server i for some bit index $W-u$ where $u \in [0, W-1]$, its state for $u+1$ can be determined based on $B^i_{W-u-1} \in \{0,1\}$, and $q^i_{W-u-1} \in \{-1,0,1\}$, where $B^i_{W-(u+1)}$ is the bit located at index $W-(u+1)$ in the binary representation of d^i, and $q^i_{W-(u+1)}$ is determined by the existence of a rule with the corresponding prefix length that involves server i. Following Theorem 6, there exists an optimal solution such that $q^i_{W-(u+1)} \in \{-1,0,1\}$.

Table V-A captures this dependency given a server that is in positive state for a bit index $W-u$. For example, given that the server is in positive-state for bit $W-u$ meaning that $d^i_{W-u} = h^i_u + 2^{W-u}$, then if $B^i_{W-(u+1)} = 0$ and $q^i_{W-(u+1)} = -1$, we get that $d^i_{W-(u+1)} = d^i_{W-u} - 2^{W-(u+1)} = h^i_{u+1} + 2^{W-(u+1)}$, that is, the server stays in a positive-state for bit index $u+1$. On the other hand, if $q^i_{W-u} = 0$, we would get neither of the states as defined. Similar tables given a negative-state and a zero-state for bit u can be obtained.

Consequentially, for each server, based on its state for bit index $W-u$ and the value of $B^i_{W-(u+1)}$, we can describe whether a rule that refers to it should be added, and if so whether this can be a positive rule (increasing $q^i_{W-(u+1)}$ by one) or a negative rule (decreasing $q^i_{W-(u+1)}$ by one). The result state of the server for bit index $W-(u+1)$ depends on this choice.

Since any distribution can be represented using the vector set $\hat{Q}$ with the constraints reflected in Section IV that is, for each prefix length there is at most one rule that changes the number of bit combinations mapped to each server, the search space for the optimal exact traffic split is bounded. Moreover, by Theorem 8, we can consider an additional constraint on the vector set $\hat{Q}$ by which for each server i and prefix length v, the (weighted) partial sum d^i_u, corresponding to the high indices (equal at least $W-u$), can take at most two values. We refer to the state of all servers for some index u as a super-state.

The suggested search algorithm considers values $u \in \{0, 1, \ldots, W-1\}$, and is defined on these superstates. Initially, we have only one superstate where all servers are in negative states except for the server who gets the default rule and is in a positive state. At each iteration, the algorithm iterates over all superstates and for each one of them, it calculates the reachable super-states, and the minimal number of rules required to achieve each such super-state. For a value $u+1$, the algorithm allows adding rules with prefix length corresponding to $q^i_{W-(u+1)}$. Since for some given v, each server can have one of three possible states, the number of super-states is clearly bounded by 3^k. In our experiments we observe that the number of reachable states is in practice often much smaller although still exponential in the number of servers.

For $u \in \{0, \ldots, W-1\}$, for the transition towards the value of $u+1$, we consider for $i \in [1, k]$ the value B^i_{W-u}, the corresponding bit in the binary representation of c^i. We calculate the set of reachable super-states for u based on those reachable for $u-1$. Given a state of a server, the value of $B^i_{W-(u+1)}$ determines the possible rule additions to that server. Given a super-state for index u and the values $B^1_{W-(u+1)}, \ldots, B^k_{W-(u+1)}$ for the various servers, we consider rule combinations for which an identical number of positive rules and negative rules are added. We calculate the achievable super-states for $u+1$, each associated with a number of required rules based on the number of rules required for the super-state for index u and the additional required rules. We keep for each super-state for index $u+1$, the minimal number of rules that can lead to it.

The pseudo-code of this algorithm is described in Algorithm 1. We use iSS, cSS and SS to stand for the initial, the current and a general SuperState, respectively. For the sake of brevity, the algorithm described only computes the optimal rule count. For computing the actual vector set $\hat{Q}$, one need to keep track for each super-state, the super-state that it was reached from. By the chain of the super-states the vector set $\hat{Q}$ can be recovered. The solution is determined from the super-states for $v = W$. In particular, to correctly represent the target distribution we need all k servers to be of a zero state. The required rules number is the count associated with this super-state. By Theorem 6 the solution can be realized, where the actual realization can be performed by Theorem 7. The minimality of the representations in each iteration implies the optimality of the algorithm.

Last, one may iterate over all $p \in \{1, \ldots, k\}$ and get the server who the default rule is applied to that leads to the optimal number of rules.

Algorithm 1: Algorithm for computing the optimal number of rules given server p gets the default rule

```
Input:  A target traffic distribution $C = (c^1, \ldots, c^k)$.
        Server $p$ who gets the default rule.
Output: An optimal number of rules realizing $C$

$iSS.count$ = 0; $iSS.state$ = all servers in negative-state,
except for server $p$ who is in positive state;
$A_0 = \{iSS\}$;
For all $i, u$, $B^i_u = u^{th}$ binary bit of $c^i$;
for $u \in \{0, 1, \ldots, W-1\}$ do
    $A_{u+1} = \emptyset$
    for $cSS \in A_u$ do
        • Find for each server optional rule additions
        • Consider balanced options to calculate the possible next
          super-state added to $A_{u+1}$
Find $SS \in A_W$ with $k$ servers in zero-state that
minimizes $SS.count$.
return $SS.count$
```

B. Approximated distribution realization

Given a restriction on the rule number, a simple approach is to take the n first added (lowest priority) rules in an optimal solution for an exact representation. However, we conclude by the following example that this approach is not optimal.

Example 2. *Consider the target distribution $C = (2, 3, 3, 8)$. In its first two rules, the solution for exact representation applies a default rule to the last server, and the next rule as a rule of size 2^3 mapping to server 3 and eliminating traffic from server 4, resulting in a maximum excess traffic $G(D)$ of 5 (to server 3). However, using two rules one can achieve a value $G(D) = 4$, by replacing the second rule in the above solution to be of size 2^2.*

Our approach is based on intuition taken from study of properties of the algorithm for optimal exact realization. Due to space constraints we provide the high level ideas. We basically follow the exact same steps as the algorithm for exact realization where we keep record of the maximum excess traffic $G(D)$ of any distribution that is encoded by each super-state the algorithm arrives at. We find for each number of rules, the super-state that minimizes the maximum excess traffic $G(D)$.

For a more accurate consideration of the super-states we arrive at, we also consider transitions between super-states that involves more than one rule, for which we carefully, in a separate sub-routine, add the involved rules one by the other, where the next rule to be added is the one that results in the minimal $G(D)$. For each such sub-step we also record the value of $G(D)$.

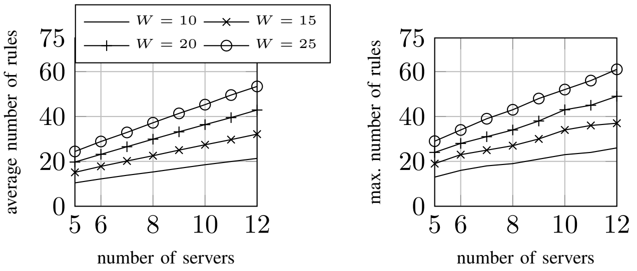
The algorithm then outputs for each number of rules the minimal $G(D)$ encountered and its corresponding distribution (along with the corresponding super-state or the corresponding sub-step of a transition between super-states).

VI. Experimental Results

A. *Effect of number of servers on exact realization size*

In this section we examine the optimal number of rules needed for an exact realization of a distribution as given by the algorithm from Section 1.

Fig. 6 shows the average and maximum optimal number of rules over 500 random traffic allocations for each $k \in \{5, 6, \ldots, 12\}$ servers and a number of bits $W \in \{10, 15, 20, 25\}$. For given values of k and W, randomization



(a) average number of rules (b) maximum number of rules

Fig. 6. Average and maximum optimal number of rules over 500 uniformly distributed traffic allocations as a function of the number of servers k and the number of bits W. Legendary refers to both subfigures.

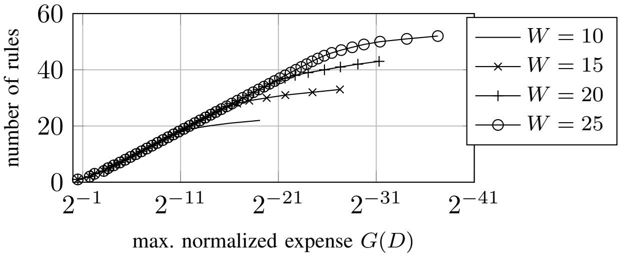


Fig. 7. Expectation of number of rules required for a maximum normalized expense $G(D)$. Results are shown for the average of 5000 random instances with $k = 10$ servers and $W \in \{10, 15, 20, 25\}$ bits.

was performed such that the traffic allocations, represented as a vector with sum 2^W, is drawn uniformly from the space of all integer vectors with sum 2^W (and all elements are non-negative). To create these random traffic allocations, that is, fixed-sum vectors, we used a result related to Dirichlet distribution [27], where first we generated $a^1, \ldots, a^k$ uniformly distributed numbers in $[0, 1]$. Then the allocation c^i for a server is given by the closest integer of $2^W \cdot \log a^i / (\sum_{j=1}^{k} \log a^j)$ with last small corrections due to rounding so that their sum is 2^W. The results show a linear increase in the rule number as a function of the number of servers.

Last, we note that in all instances that we tested we obtained the exact same optimal number of rules as Niagara [12], although the later one is not proven to be optimal.

B. *Approximate realization of single flow*

We now investigate the number of rules needed to achieve a given normalized maximum amount of expense $G(D)$. We used the same method from Section VI-A for creating uniformly distributed fixed sum target traffic distribution. Fig. 7 shows the average number of required rules in a solution found by our algorithm for approximating traffic allocation for a given (normalized) maximum allocation expense $G(D)$. The results are based on the average of 5000 random instances with $k = 10$ servers and $W \in \{10, 15, 20, 25\}$ bits. Interestingly, the number of required rules grows linearly with the logarithm of the maximum allocation expense $G(D)$ in a rate similar for the various bit numbers. This is because our algorithm deals first with the most significant bit and then considers lower bits, making it indifference to the actual number of bits.

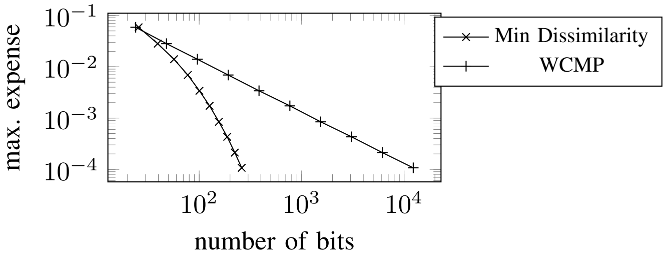


Fig. 8. Relation between the allocated memory and the maximum expense for our approach and WCMP, with $k = 10$ servers.

C. *Comparison with WCMP*

We compare our approach with WCMP [8]. In this approach, an array maintains mapping values with various multiplicities. One array entry is accessed with the uniform distribution, implying an output distribution D based on the multiplicities. We created 1000 random target distributions of size $k = 8$ (servers). Then, we apply WCMP with various memory capacity values as well as our technique with various header number of bits W. We measured the normalized maximum excess traffic among servers, namely $\frac{1}{2^W} \cdot \max_{i \in [1,k]} \left(d^i - c^i \right) = \frac{1}{2^W} \cdot G(D)$ for D.

Note that while in our approach, each memory entry is of $(W + \log_2 k)$ bits, composed of W matching bits and additional $\log_2 k$ for the server index, in WCMP each entry has only a server index of $\log_2 k$ bits. Accordingly, to compare the approaches we compare their amount of total memory and not the number of entries. It is important to mention that while the WCMP can be implemented in SRAM, our approach requires the combination of TCAM and SRAM, so a memory bit for our approach can more expensive based on the implementation. Fig. 8 shows the average maximum expense as a function of average total number of bits for both techniques. Our technique outperforms WCMP in terms of accuracy given limited memory in all test cases that we have examined. For instance, to achieve a maximum average expense of 0.001, WCMP uses approximately 1500 memory bits. Our approach uses as little as approximately 156 bits.

VII. Conclusions and Future Work

In this paper, we studied the representation of traffic distributions in commodity switches. We explained the tight connection of the problem to signed representations of positive integers. This observation allows us to construct representations with optimality guarantees. As a future work, we would like to find also optimal limited size representations with a minimal error. We would also like to examine whether this link can help to understand more the expressiveness of switch memory for other typical tasks such as traffic measurement and policy enforcement.

References

[1] O. Rottenstreich, Y. Kanizo, H. Kaplan, and J. Rexford, "Accurate traffic splitting on commodity switches," in *ACM SPAA*, 2018.

[2] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. G. Greenberg, D. A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu, C. Kim, and N. Karri, "Ananta: Cloud scale load balancing," in *ACM SIGCOMM*, 2013.

[3] R. Gandhi, H. H. Liu, Y. C. Hu, G. Lu, J. Padhye, L. Yuan, and M. Zhang, "Duet: Cloud scale load balancing with hardware and software," in *ACM SIGCOMM*, 2014.

[4] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, "Let It Flow: Resilient asymmetric load balancing with flowlet switching," in *USENIX NSDI*, 2017.

[5] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "CONGA: Distributed congestion-aware load balancing for datacenters," in *ACM SIGCOMM*, 2014.

[6] C. Hopps and D. Thaler, "Multipath issues in unicast and multicast next-hop selection," Nov. 2000, RFC 2991.

[7] C. Hopps, "Analysis of an equal-cost multi-path algorithm," Nov. 2000, RFC 2992.

[8] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "WCMP: Weighted cost multipathing for improved fairness in data centers," in *ACM EuroSys*, 2014.

[9] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "SilkRoad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *ACM SIGCOMM*, 2017.

[10] T. Mizrahi, O. Rottenstreich, and Y. Moses, "TimeFlip: Using timestamp-based TCAM ranges to accurately schedule network updates," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 849–863, 2017.

[11] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-based server load balancing gone wild," in *USENIX Hot-ICE*, 2011.

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

[13] O. Rottenstreich and J. Tapolcai, "Optimal rule caching and lossy compression for longest prefix matching," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 864–878, 2017.

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

[15] S. Suri, T. Sandholm, and P. R. Warkhede, "Compressing two-dimensional routing tables," *Algorithmica*, vol. 35, no. 4, pp. 287–300, 2003.

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

[17] G. J. Narlikar, A. Basu, and F. Zane, "CoolCAMs: Power-efficient TCAMs for forwarding engines," in *IEEE INFOCOM*, 2003.

[18] S. Kasnavi, V. C. Gaudet, P. Berube, and J. N. Amaral, "A hardware-based longest prefix matching scheme for TCAMs," in *IEEE International Symposium on Circuits and Systems*, 2005, pp. 3339–3342.

[19] 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.

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

[21] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, 2006.

[22] N. J. A. Sloane and S. Plouffe, "The Encyclopedia of Integer Sequences," 1995.

[23] W. Bosma, "Signed bits and fast exponentiation," *Journal de théorie des nombres de Bordeaux*, vol. 13, no. 1, pp. 27–41, 2001.

[24] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[25] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for Internet applications," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 17–32, 2003.

[26] P. Ganesan and G. S. Manku, "Optimal routing in Chord," in *ACM-SIAM SODA*, 2004.

[27] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, 2010.

Ori Rottenstreich is a faculty member at the department of Computer Science and the department of Electrical Engineering of the Technion, Haifa, Israel. His main research interest is computer networks. In 2015-2017 he was a Postdoctoral Research Fellow at the department of Computer Science, Princeton university. Earlier, he received the BSc in Computer Engineering (summa cum laude) and PhD degree from the Technion in 2008 and 2014, respectively.

Yossi Kanizo received both the BSc degree in computer engineering and the PhD degree from the computer science department of the Technion, Haifa, Israel in 2006 and 2014, respectively. He is currently a lecturer at the Computer Science Department, Tel Hai Academic college, Israel. His main research interests are software defined networking, hash-based data-structures, and switch architectures.

Haim Kaplan received his Ph.D. degree from Princeton University at 1997. He was a member of technical stuff at AT&T research from 1996 to 1999. Since 1999 he is a Professor in the School of Computer Science at Tel Aviv University. His research interests are design and analysis of algorithms and data structures.

Jennifer Rexford is the Gordon Y.S. Wu Professor of Engineering and the Chair of Computer Science at Princeton University. Before joining Princeton in 2005, she worked for eight years at AT&T Labs–Research. Jennifer received her BSE degree in electrical engineering from Princeton University in 1991, and her PhD degree in electrical engineering and computer science from the University of Michigan in 1996. She is co-author of the book "Web Protocols and Practice" (Addison-Wesley, May 2001). She served as the chair of ACM SIGCOMM from 2003 to 2007. Jennifer was the 2004 winner of ACM's Grace Murray Hopper Award for outstanding young computer professional. She is an ACM Fellow (2008), and a member of the American Academy of Arts and Sciences (2013) and the National Academy of Engineering (2014).