Minimizing Range Rules for Packet Filtering Using Double Mask Representation

Ahmad Abboud, Abdelkader Lahmadi, Michaël Rusinowitch, Miguel Couceiro, Adel Bouhoula, Saif El Hakk Awainia, Mondher Ayadi

Minimizing Range Rules for Packet Filtering Using Double Mask Representation

Ahmad Abboud[†*], Abdelkader Lahmadi[*], Michael Rusinowitch[*], Miguel Couceiro[*]
Adel Bouhoula[‡†], Saif El Hakk Awainia[†], Mondher Ayadi[†]
[*] Université de Lorraine, CNRS, Inria, Loria, F-54000 Nancy, France, {firstname.lastname}@inria.fr
[†] NUMERYX, France, a.abboud@numeryx.fr, a.bouhoula@numeryx.fr, s.awainia@numeryx.fr, m.ayadi@numeryx.fr
[‡] Digital Security Research Lab, Sup'Com, University of Carthage, Tunisia, adel.bouhoula@supcom.tn

Abstract—Packet filtering is widely used in multiple networking appliances and applications, in particular, to block malicious traffic (protect network infrastructures through firewalls and intrusion detection systems) and to be deployed on routers, switches and load balancers for packet classification. This mechanism relies on the packet's header fields to filter such traffic by using range rules of IP addresses or ports. However, the set of packet filters has to handle a growing number of connected nodes and many of them are compromised and used as sources of attacks. For instance, IP filter sets available in blacklists may reach several millions of entries, and may require large memory space for their storage in filtering appliances. In this paper, we propose a new method based on a double mask IP prefix representation together with a linear transformation algorithm to build a minimized set of range rules. We define formally the double mask representation over range rules and we prove that the number of required masks for any range is at most $2w - 4$, where w is the length of a field. This representation makes the network more secure, reliable and easy to maintain and configure. We define formally the double mask representation over range rules. We show empirically that the proposed method achieves an average compression ratio of 11% on real-life blacklists and up to 74% on synthetic range rule sets. Finally, we add support of double mask into a real SDN network.

I. Introduction

Multiple network appliances and applications including firewalls, intrusion detection systems, routers, and load balancers rely on a filtering process using sets of rules to decide whether to accept or deny an incoming packet. Effective filtering is essential to handle the rapidly increasing and the dynamic nature of network traffic where more and more nodes are connected, due to the emergence of 5G networks and the increasing number of sources of attack.

With the large number of hosts, it remains crucial to minimize the number of entries in routing tables and to accelerate the lookup process. According to [1], current routing tables contain more than 600k entries, this number will surpass 1 Million in 2020.

Furthermore, when simulating large-scale networks using commodity hardware, the size of routing table is a hard constraint due to the limited resources of the used computers [2]. The problem of minimizing the size of the routing table is also present in Software-Defined Networks.

On the other hand, according to [3], a routing table that uses Border Gateway Protocol (BGP), may have more than 500k entries and needs to be installed into OpenFlow switches that use ternary content addressable memories (TCAM) to store them for fast packet classification. However, TCAM has a limited capacity, a high power consumption and a high cost [4].

On the other hand, attacks on Internet have reached a high level according to [5]. The rate of spam mail has reached 53% in 2016 and more than 229000 web attacks have been detected each day. The number keeps increasing which in turn increases the size of blacklists and the number of rules in firewalls. The limited storage capacity [6] requires efficient management of that space.

To face the large number of hosts and routing tables, [7] developed Classless Inter-Domain Routing (CIDR) to replace the classful network architecture. The CIDR allows prefixes of arbitrary lengths. This notation relies on a prefix and a mask as follows: $a.b.c.d/m$. The prefix $a.b.c.d$ identifies the network, and the mask m identifies hosts in the network. However, using this notation to represent routing table rules that contain ranges can lead to multiple entries and thus there is a need for a better notation along with an efficient algorithm to reduce the number of entries and therefore the classification and lookup time, and memory usage [8].

Reducing range rule sets for classification or filtering, has been extensively studied in the literature. The main goal in these studies is to reduce the number of entries by mainly using optimization techniques while keeping their intended semantics. However, all these minimization proposals still rely on the notions of single prefix or mask, in particular representation of filtering rules. Some previous works have provided solutions to the packet classification problem by introducing algorithms for fast packet filtering [9]. Other works have focused on reducing the size of routing table by removing the redundant rules [6], [10], [11].

In this work, our main goal is to find a simple representation of filtering rules that allows one to get more compact rule tables, easy to manage, whilst keeping their semantics unchanged. The construction of rules should be obtained with reasonably efficient algorithms too.
To achieve this goal, we introduce a novel representation of packet filter fields, so called double masks [12], where

the first mask is used as an inclusion prefix and the second as an exclusion one. This new representation can add flexibility and efficiency in the deployment of security policies, since the generated rules are easier to manage. The double mask representation makes configurations simpler since we can accept and exclude IPs within the same rule. A double mask rule can be viewed as an extension of a standard prefix rule with exceptions. It is often more intuitive than alternative representations and therefore can prevent errors in network management operations.

In this paper, we demonstrate the practicality of this representation over range rules and we prove that the number of required masks for any range is at most $2w-4$, where w is the length of a field. We provide an efficient algorithm (linear time) that is able to build the double mask representation of a set of range rules.

To summarize, our contributions are threefold:

1) We formally define the double mask representation over range rules.
2) We design a linear algorithm to transform range rules into a double mask representation.
3) We empirically show that the proposed algorithm achieves an average compression ratio of 11% on real-life blacklists and up to 74% on synthetic range rule sets.

The remainder of this paper is organized as follows. In Section II, we formally introduce the double mask representation. In Section III we propose an algorithm to compute the masks for a given range using double and simple mask representations. In Section IV we show that $2w-4$ masks are sufficient for representing any range, and that for some ranges we cannot do better. Section V presents a more efficient linear algorithm for building double mask representations over range fields. In Section VI, we describe our experiments and the performance evaluation results of the proposed algorithm using real-life and synthetic datasets. In Section VII, we present the hardware implementation for adding support of the double mask representation in SDN networks and the matching process between a packet and a double mask rule. Related works are discussed in Section VIII. In Section IX we present the conclusion and discuss topics of future research.

II. Double Mask Representation

A. Notation and definitions

Before introducing the double mask representation, we define the notation used throughout the paper, that is summarized in Table I.

TABLE I: Notation employed throughout the paper.

ip	IP address
w	number of bits representing an IP adress
P	prefix covering an ip
$bin_v(a)$	binary representation of integer a using v bits
$val_v(a)$	integer value of bitstring a with length v
range $[a,b]$	set of IP addresses with value between a and b
t_w	perfect binary tree of height w
ε	empty bitstring
p	bitstring (or path in t_w)
$\|p\|$	length of p
$t(p)$	perfect binary subtree of t_w with root p
$l(p)$	set of leaves of $t(p)$
$p\|k$	prefix of length k of p
SM_p	set of simple masks covering $l(p)$
DM_p	set of double masks covering $l(p)$

1) Prefix, Simple mask: A prefix P is a word of length w on alphabet $\{0,1,*\}$ where all $*$'s occur at the end of the word: $P = p_{k-1}...p_0*^i$ where $k+i=w$. To avoid confusion with the usual notion of word prefix, we will also sometimes call P a simple mask. An address $ip \in \{0,1\}^w$ is covered by simple mask P if $ip_i = p_i \ \ for\ i \in [k, k-i+1]$. The subword $p = p_{k-1}...p_0$ is called the path of P for reasons to be explained below.

2) Range: A range is denoted by an integer interval $[a,b]$ (where $0 \le a \le b \le 2^w-1$). A range represents the set of IP addresses ip, with integer value $val_w(ip)$ between a and b.

3) Perfect Range: $[a,b]$ is a perfect range if there is $r \in \{0,1\}^{w-k}$ such that $bin_w(a) = r.0^k$ and $bin_w(b) = r.1^k$.

4) Perfect Binary Tree: Note that the IP addresses of length w are in bijection with the leaves of a perfect binary tree t_w of height w. More generally, we can define the following bijection $t()$ on the set of bitstrings of length $\le w$ with the perfect binary subtrees of t_w : $t(\varepsilon) = t_w$ where ε is the empty bitstring, and given bitstring v we define $t(v0)$ (resp. $t(v1)$) to be the left (resp. right) subtree of $t(v)$. In particular, if prefix P has a path p of length k the IP addresses covered by P are exactly the leaves of the perfect subtree $t(p)$. This set of addresses is a perfect range. In fact, every perfect range is also the set of leaves of a perfect subtree.

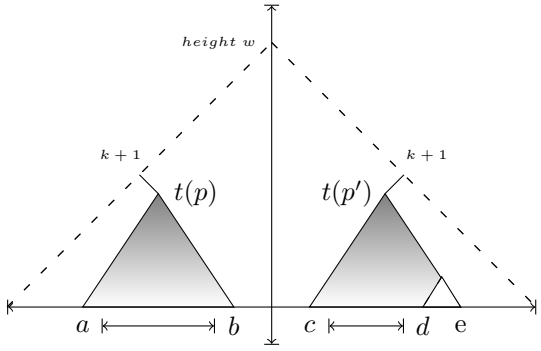

Fig. 1: Illustration of a perfect Binary Tree.

In Fig. 1, $[a,b]$ is a perfect range as it is the set of leaves of the perfect binary tree $t(p)$ of height k. However, $[c,d]$ is not a perfect range as it is not the set of leaves of a perfect binary tree.

B. Definition of the double mask representation

The double mask representation has been introduced informally in [12] in order to reduce the number of filtering rules. We now define this representation for range fields, in particular for IP address fields. Note that the representation can be applied to other range fields such as ports.

We assume in the following that IP addresses are binary words of length w, i.e., IP addresses are elements of $\{0,1\}^w$ indexed from 1 to w. A double mask representation has three components and is denoted $netpref/mask1/mask2$. The first component $netpref \in \{0,1\}^w$ is a network prefix. The second and third components are integers $mask1, mask2 \in [0,w]$. Component $mask1$ defines all accepted IPs, and component $mask2$ defines all excluded IPs from the list of accepted ones.

Definition 1. An IP address ip is in the set defined by $netpref/mask1/mask2$ if $ip_i = netpref_i \ for \ i \in [1,\ldots,mask1]$ and there exists $j \in [mask1+1, mask2]$ such that $ip_j \neq netpref_j$. In that case we say that ip is covered by $netpref/mask1/mask2$.

Let us consider the following example of a double mask representation:

$$192.168.100.96/26/2$$

This representation means that any selected (or filtered) adress must have its 26 first bits equal to the 26 first bits of 192.168.100.96 (that is equal to 11000000.10101000.01100100.01), and at least one of the 2 following bits 27,28 should not be equal to the corresponding bit of 192.168.100.96. In other words either bit 27 is not 1 or bit 28 is not 0. As we will see, this new representation can reduce the number of filtering rules dramatically. It is also possible to represent more explicitly the double mask as a word where the forbidden combination of bits is overlined, the leftmost part specifies the fixed bits and the rightmost part the free bits (that are allowed to take any value). The two possible notations of a double mask are given below:

$$N1: \quad a_{k-1}...a_0 a_{j-1}...a_0 0_{i-1}...0_0/k/j$$

$$N2: \quad a_{k-1}...a_0 \overline{a_{j-1}...a_0} 0_{i-1}...0_0$$

where $(i+j+k=w)$, and if $j=0$ the double mask is equivalent to a simple mask (or a TCAM entry) $a_{k-1}...a_0 *^{w-k}$.

When designing filtering rules, it is useful and more efficient to represent the excluded addresses rather than the accepted ones, especially, when there are much more excluded addresses than accepted ones.

In this case using a double mask representation has a better effect, since by reducing the number of filtering rules, we reduce computation time, memory and power usage.

The examples below illustrate the benefits of using double masks over simple masks.

Example 1. Range [1,14] needs a set of 6 standard prefixes to be represented. However this range can be represented using only two double masks prefixes as shown below :

$$\begin{array}{lll} \text{range} & \text{simple masks} & \text{double masks} \\ [1,14] = & \begin{cases} 0001 \\ 001* \\ 01** \\ 10** \\ 110* \\ 1110 \end{cases} & \begin{cases} \overline{0000} \\ \overline{1111} \end{cases} \end{array}$$

Example 2. Range $[1,15]$ is of form $[1,2^4-1]$ and needs 4 simple masks $\{0001, 001*, 01**, 1***\}$ but only one double mask: $\overline{0000}$.

More generally, a range $[1,2^w-1]$ can be represented by a unique double mask $\overline{0}^w$ but cannot be represented by less than w simple masks. Let us demonstrate this by contradiction. Let us assume that $[1,2^w-1]$ can be represented by strictly less than w simple masks. Then at least two different addresses $2^i-1, 2^j-1 (j>i)$ are covered by the same mask. The mask has to be a common prefix of their binary representations: therefore it has to be a prefix of 0^{w-j}. However, in that case, the mask would also cover 0^w, which is a contradiction.

III. Double Mask Computation

We now present an algorithm to generate a set of double masks that covers a range $[a,b]$, i.e., selects exactly the addresses in this range.

The algorithm proceeds recursively on the binary tree $t_w = t(\varepsilon)$ that stores all IP addresses of size w. Note that each node of $t(\varepsilon)$ can be located uniquely by a path (bitstring) p from the root to this node: the root is located by ε; the left and right child of the node located by p are located by $p0$ and $p1$ respectively. We will identify a node with the path that locates it. A path can also be viewed as a prefix where the $*$'s are omitted. The leaves of $t(\varepsilon)$ are the IP addresses. We denote the set of leaves of subtree $t(p)$ by $l(p)$. Algorithm computes in a bottom-up way a set of double masks covering $l(p)$. Moreover we denote these partial results by by DM_p. We denote by $\bar{i}$ the complement of boolean i, i.e., $\bar{0}=1$, $\bar{1}=0$.

To process a node p in $t(\varepsilon)$ we have to consider several cases according to the left and right children of p, as described below and as illustrated in Fig. 2.

Case 0: if p is a leaf and $l(p) \subseteq [a,b]$, then $DM_p = \{p/|p|/0\}$, else $\emptyset$

Case 1: if $l(p0)$ and $l(p1)$ are both subsets of $[a,b]$ then $DM_p = \{p0^{w-|p|}/|p|/0\}$

Case 2: if there is a unique $i \in \{0,1\}$ such that $l(pi)$ is a subset of $[a,b]$ then

Case 2.1: if $DM_{p\bar{i}} = \{p\bar{i}dq/|p|+2/0\}$ $(d \in \{0,1\})$ then
$DM_p = \{p\bar{i}\bar{d}q/|p|/2\}$
Case 2.2: if $DM_{p\bar{i}} = \{p\bar{i}q/|p|+1/m\}$ (where $m > 0$) then
$DM_p = \{p\bar{i}q/|p|/m+1\}$
Case 3: Otherwise $DM_p = DM_{p0} \cup DM_{p1}$

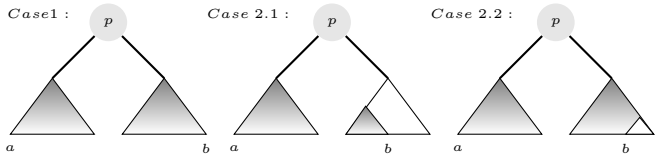

Fig. 2: Typical examples for *Cases* 1, 2.1 *and* 2.2

Algorithm 1 NaiveMasks(a,b)

```
1: Input: a,b
2: Output: set of double masks representing [a,b]
3: return DM_ε where:
4: if p is a leaf then
5:    if p ∉ [a,b] then
6:       return DM_p = ∅
7:    else
8:
9:       return DM_p = {p/|p|/0}                          ▷Case 0
10:   end if
11: end if
12: if l(p0), l(p1) ⊆ [a,b] then
13:
14:   return DM_p = {p0^{w-|p|}/|p|/0}                    ▷Case 1
15: else
16:   if l(pi) ⊆ [a,b] then
17:      if DM_{pī} = {pīdq/|p|+2/0} (d ∈ {0,1}) then
18:
19:         return DM_p = {pīd̄q/|p|/2}                    ▷Case 2.1
20:      else
21:         if DM_{pī} = {pīq/|p|+1/m}  (m > 0) then
22:
23:            return DM_p = {pīq/|p|/m+1}                ▷Case 2.2
24:         end if
25:      end if
26:   end if
27: end if
28:
29: return DM_p = DM_{p0} ∪ DM_{p1}                       ▷Case 3
```

Given a range $[a,b]$, the set of double masks DM_ε returned by Algorithm 1 covers $[a,b]$. The proof of correctness is to demonstrate by induction on $w-|p|$ that DM_p spans $l(p) \cap [a,b]$. For the base case $|p| = w$ and $l(p)$ is an IP address: then $DM_p = \{p/0/0\}$. For the induction step we have to prove by cases that if DP_{pi} spans $l(pi) \cap [a,b]$ and $DP_{p\bar{i}}$ spans $l(p\bar{i}) \cap [a,b]$ then DM_p spans $l(p) \cap [a,b]$. We then conclude that DM_ε spans $l(\varepsilon) \cap [a,b] = [a,b]$.

Fig. 3 gives an illustrative example of the algorithm execution.

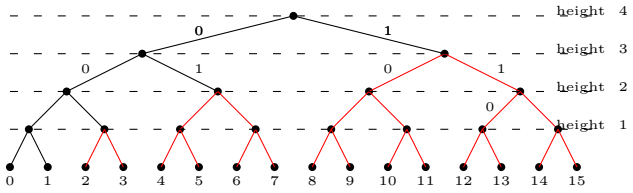

Fig. 3: Example : Let $[a,b] = [2,15]$. The algorithm start from the button. $[2,2]$ is a leaf and $\in [a,b]$. According to Case 0, $DM_{0010} = \{0010/1/0\}$. Same for each leaf in $[a,b]$. At height 1, $l(p0), l(p1) \subseteq [a,b]$. The algorithm return $DM_p = \{p0^{w-|p|}/|p|/0\}$ according to Case 1. For $[2,3]$, $l(p0) = 2$ and $l(p1) = 3$. In this case, $DM_{001} = \{0010/3/0\}$. At height 2, according to Case 3, $DM_{00} = DM_{000} \cup DM_{001}$, but $DM_{000} = \emptyset$ since $0,1 \notin [a,b]$, so $DM_{00} = \{0010/3/0\}$. For $[4,7], [8,11]$ *and* $[12,15]$, $l(p0)$, $l(p1) \subseteq [a,b]$. According to the algorithm, $DM_p = \{p0^{w-|p|}/|p|/0\}$. For example, in $[4,7]$, $l(010)$, $l(011) \subseteq [2,15]$, the algorithm return $DM_{01} = \{0100/2/0\}$. At height 3, for $[2,7]$, $l(00) \notin [2,15]$ but $l(01) \in [2,15]$ and $DM_{00} = \{p\bar{i}dq/|p|+2/0\} = \{0010/3/0\}$. DM_0 will be equal to $\{0000/1/2\}$ according to Case 2.1. For $[8,15]$ the algorithm return $DM_1 = \{1000/1/0\}$ since $l(10)$, $l(11) \subseteq [2,15]$. At height 4, $l(0) \notin [2,15]$, but $DM_0 = \{0000/1/2\}$, according to Case 2.2 the algorithm return $DM = \{0000/0/3\}$.

IV. Bounding the number of masks to represent a range

In this section we show that the number of masks needed to represent a range in Algorithm 1 is at most $2w-4$. Moreover we will show that this upper bound is tight.

Proposition 1. Let $v \geq 2$ and $0 \leq a,b \leq 2^v-1$. Any range of type $[a,2^v-1]$ or $[0,b]$ can be represented by at most $v-1$ masks.

Proof: By symmetry we only consider $[a,2^v-1]$. We perform an induction on v. If $bin_v(a) = 0^k1s$ with $k \leq v-2$. By induction hypothesis applied to $v-k-1$ $[val_v(s), 2^{v-k-1}-1]$ can be represented by $v-k-1$ masks in $t(0^k1)$. These masks can be extended to masks in v bits by adding 0^k1 to the left of the network prefix and adjusting the masks components. The complement $[2^{v-k-1}, 2^v-1]$ can be represented by the k prefixes $01*^{v-2}, 0^21*^{v-3}, \ldots, 0^{k-1}1*^{v-k}$. Overall we get $(v-k-1)+k = v-1$ masks. If $a = 0^{v-1}1$ then $[a,2^v-1]$ is represented by a double mask excluding 0. If $a = 0^v$ then $[a,2^v-1]$ is represented by a simple mask associated to prefix $*^v$. Hence the proposition holds. □

Using Proposition , we will show now that, for $w > 2$, any range can be covered with at most $2w-4$ double masks.

Proposition 2. Let $w > 2$. Every range $[a, b] \subseteq [0, 2^w - 1]$ can be represented by at most $2w - 4$ masks.

Proof: Let $[a, b]$ be a range of addresses of length w. It is well known, according to [13], that $n \leq 2w - 2$ simple masks are sufficient to represent a range $[a, b] \subseteq [0, 2^w - 1]$.

Now let us prove by induction that a range $[a, b]$ of w bits can be represented with $\leq 2w - 4$ masks.
For $w = 3$, two masks are sufficient. For $w = 4$, four masks are sufficient to represent any range $[a, b]$.
Assume the proposition holds for $w - 1$ bits. We perform a case analysis and assume that one case is applied only if the previous ones are not applicable:

- $[a, b] \subseteq l(0)$ or $[a, b] \subseteq l(1)$ then by induction hypothesis the proposition holds.
- $[a, b] \subseteq l(01) \cup l(10)$ then applying Proposition IV to $[a, b] \cap l(0)$ with $v = w - 2$ we obtain that $[a, b] \cap l(0)$ is covered by $v - 3$ masks. These masks can be extended to masks in w bits. In the same way $[a, b] \cap l(0)$ is covered by $w - 3$ masks. Therefore $[a, b]$ can be represented with $2w - 6$ masks.
- $[a, b] \subseteq l(01) \cup l(10) \cup l(11)$ we apply the previous item reasoning to show that $[a, b] \cap (l(01) \cup l(11))$ is represented by $2w - 6$ masks. Since one mask is sufficient for perfect range $[a, b] \cap l(10)$, we obtain overall $2w - 5$ masks.
- $[a, b] \subseteq l(00) \cup l(01) \cup l(10)$: we reason as in the previous case.
- $[a, b] \subseteq l(00) \cup l(01) \cup l(10) \cup l(11)$: we need $2w - 6$ masks for $[a, b] \cap (l(00) \cup l(11))$, one mask for each of $[a, b] \cap l(01)$ and $[a, b] \cap l(10)$ since they are perfect ranges. Hence overall $2w - 6 + 2 = 2w - 4$ masks are sufficient.

Therefore the total number of masks needed to represent $[a, b]$ is $2w - 4$. □

The following proposition shows that the $2w - 4$ bound is tight, i.e., some ranges cannot be represented by less than $2w - 4$ double masks.

Proposition 3. Let $w > 3$. The range $[3, 2^w - 4]$ cannot be represented by less than $2w - 4$ double masks.

Proof: First note that no mask can cover a set of addresses with non empty intersection with both $l(0)$ and $l(1)$. Therefore we have to add the minimal number of masks for covering $[3, 2^{w-1} - 1]$ and the minimal number of masks for covering $[2^{w-1}, 2^w - 4]$. Address 3 cannot be covered by a mask $s/p/k$ with $p < w - 1$: otherwise, if $k > 0$ only a unique perfect subrange $l(s|p + k)$ would be excluded, but $[0, 2]$ is composed of two perfect subranges, contradiction ; if $k = 0$ then $l(s|p)$ would contains address 2, contradiction. Hence address 3 can be covered only by $0^{w-2}11/w/0$ or $0^{w-1}10/w - 1/1$. In the same way, no address between 3 and $2^{w-1} - 1$ can be covered by a double mask. By reasoning as in Example 2 we can also show that two addresses of type $2^{w'-1} - 1$ with $3 < 2^{w'-1} - 1 \leq 2^{w-1} - 1$ cannot be covered by the same simple mask. As a consequent the minimal number of masks needed to cover $[3, 2^{w-1} - 1]$ is $w - 2$. By symmetry this is also true for $[2^{w-1}, 2^w - 4]$. The total number of masks is therefore $2w - 4$. □

V. Linear time algorithm

In this section we introduce a more efficient algorithm, named $DoubleMasks$, to compute a set of masks covering any range $[a, b]$. As it will become clear, this algorithm is linear in k where k is the number of bits to represent an IP address. Given two binary strings u, v we write $u \prec v$ (resp. $u \preceq v$) when u is a strict prefix (resp. prefix) of v. We denote by $prec(p)$ the longest proper suffix of p. Recall that $u < v$ indicates that the natural number denoted by u is smaller than the one denoted by v. We assume that $bin_w(a) = ca', bin_w(b) = cb'$ where c is the longest common prefix of $bin_w(a)$ and $bin_w(b)$.

$NaiveMasks$ (Algorithm 1) processes all nodes on paths from the root to leaves with value in $[a, b]$.Hence the number of processed nodes can be exponential in w. Unlike $NaiveMasks$, the second proposed algorithm $DoubleMasks$ (Algo. 3) only processes nodes p in paths leading to leaves with value a or b, i.e., $DoubleMasks$ examines only two branches in the tree $t(\varepsilon)$. Fig. 4 shows us the idea behind the algorithm. $DoubleMasks$ works in two phases. The algorithm computes first for each node p a set of masks for $l(p) \cap [a, b]$, in a bottom up way and starting from the two nodes $bin_w(a)$ and $bin_w(b)$. Then, when reaching node c, the set of masks computed at the siblings of c (i.e., $c0$ and $c1$) are combined and the algorithm stops. This strategy is justified by the following Fact 1:

Fact 1. Let c be the longest common prefix of $bin_w(a)$ and $bin_w(b)$. Interval $[a, b]$ is the disjoint union of $[a, val_w(c01^{w-|c|-1})]$ and $[val_w(c10^{w-|c|-1}), b]$.

Now we introduce $ComputeMasks$, a procedure that computes the double-mask DM representation of each subinterval in Fact 1. $ComputeMasks$ has a parameter x that will be successively substituted by a and b in the main algorithm $DoubleMasks$. The Boolean parameter β is chosen such that $c\beta$ is a prefix of x. If $x < val_w(c\beta\bar{\beta}^{w-|c|-1})$ (resp. $x > val_w(c\beta\bar{\beta}^{w-|c|-1})$), the algorithm computes a DM representation of range $[x, val_w(c\beta\bar{\beta}^{w-|c|-1})]$ (resp. $[val_w(c\beta\bar{\beta}^{w-|c|-1}), x]$). ComputeMasks relies on the following case analysis:

Case 1: $c \prec p\beta \preceq x$:

- Case 1.1: if $DM_{p\beta} = \{p\beta\bar{\beta}s/|p| + 2/0\}$, then $DM_p = \{p\beta\beta s/|p|/2\}$ (DM generated)
- Case 1.2: if $DM_{p\beta} = \{p\beta s/|p| + 1/k\}$, then $DM_p = \{p\beta s/|p|/k+1\}$, since $l(p\bar{\beta}) \subseteq [a, b]$ (DM extended)

Case 1.3: if $DM_{p\beta} = \{p\beta s/|p|+1/0\}$, then $DM_p = \{p\beta s/|p|/0\}$, since $l(p\bar{\beta}) \subseteq [a,b]$ and $DM_{p\beta}$ is a simple mask (SM Extended)

Case 1.4: otherwise $DM_p = DM_{p\beta} \cup \{p\bar{\beta}s/|p|+1/0\}$, since $l(p\bar{\beta}) \subseteq [a,b]$ (SM added)

Case 2: if $c \prec p\bar{\beta} \preceq x$, then $DM_p = DM_{p\bar{\beta}}$ (masks maintained)

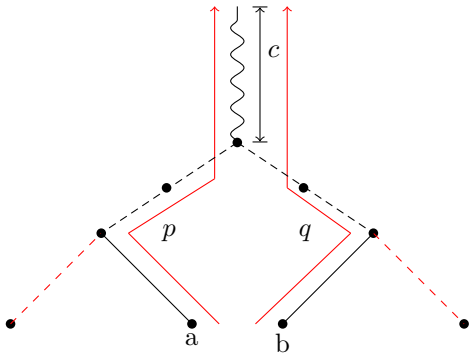

Fig. 4: Illustration of *DoubleMasks* strategy.

Now we detail the auxiliary procedure *ComputeMasks* and the main procedure *DoubleMasks*.

1) *ComputeMasks* (Algo. 2): This algorithm takes a binary number x such that $c\beta \prec x$ and returns a set of masks representing the interval between x and $val_v(c\bar{\beta}^{w-|c|})$.

First we add the simple mask corresponding to x (line 3). Then, we proceed on all prefixes of x from the longest one (lines 4-20). For each prefix, the algorithm checks the type of the previously computed mask. If this mask contains a $mask1$ of length $|p|+2$ (corresponding to a perfect tree of height $|p|+2$) a new double mask is generated (lines 7-8). If a double mask is present, this double mask will be extended (lines 9-10). If the mask computed before has a $mask1$ of length $|p|+1$ the same mask will be extended (lines 11-12). If neither of the previous cases holds, the algorithm adds a new mask to the set of masks computed before (lines 13-14).

The proof of correctness of *ComputeMasks* is by induction on $w-|p|$ and checks whether DM_p covers $l(p) \cap [a,b]$, as for NaiveMasks. Therefore the result of $ComputeMasks(a,c,0)$ (resp. $ComputeMasks(b,c,0)$) is a DM representation of $l(c0) \cap [a,b]$ (resp. $l(c1) \cap [a,b]$).

2) *DoubleMasks* (Algo. 3): This algorithm takes as input an interval $[a,b]$ and computes a set of masks representing it. First, the algorithm computes the common prefix c of a and b (line 3). Then, according to Fact 1, interval $[a,b]$ can be divided into $[a, val_w(c01^{w-|c|-1})]$ and $[val_w(c10^{w-|c|-1}), b]$. *ComputeMasks* is called for each subinterval (lines 5-6). The final result depends on the sets of masks DM_p and DM_q generated by Algo. 2. If $l(c0), l(c1) \subseteq [a,b]$ a new simple mask is generated (lines 7-9). If a double mask is generated for $t(c1)$ (resp. $t(c0)$) , the double mask will be extended (lines 10-11) (resp. 18-19). If $l(c0)$ (resp. $l(c1)$) is covered by a simple mask of length $|p|$ (resp. $|q|$) and $l(c1)$ (resp. $l(c0)$) is covered by a mask of length $|q|+1$ (resp. $|p|+1$), then a new double mask will be generated (lines 12-13) (resp. lines 20-21). If not, the algorithm returns the union of the two parts.

Algorithm 2 ComputeMasks(x,c,β)

```
1: Input: x, c, β such that cβ ≺ x
2: Output: set of masks DM_cβ
3: p ← prec(x); DM_x ← {x/w/0}        ▷processing path x
4: while c ≺ p do
5:    if case 1 then
6:       switch (DM_pβ)
7:       case 1.1 = {pββ̄s/|p|+2/0}:
8:          DM_p ← {pββs/|p|/2}        ▷new DM generated
9:       case 1.2 = {pβs/|p|+1/k}:
10:         DM_p ← {pβs/|p|/k+1}        ▷DM extended
11:      case 1.3 = {pβs/|p|+1/0}:
12:         DM_p ← {pβs/|p|/0}        ▷SM extended
13:      default:
14:         DM_p ← DM_pβ ∪ {pβ̄β^{|w|-|p|-1}/|p|+1/0} ▷SM
            added - Case 1.4
15:      end switch
16:   else
17:      case 2
18:   end if
19:   p ← prec(p)        ▷process parent node on the path
20: end while
21: return DM_p
```

DoubleMasks computes a DM representation of $l(c) \cap [a,b]$ from DM representations of $l(c0) \cap [a,b]$ and $l(c1) \cap [a,b]$ obtained by calling *ComputeMasks*. We can stop when reaching c in the main "while" loop of *DoubleMasks* and return the result DM_c since we can see easily that $l(c) \cap [a,b] = l(\varepsilon) \cap [a,b] = [a,b]$.

VI. Experimental Results

In this section we evaluate the performance of the Algorithm *DoubleMasks* and we compare it with the algorithm that only generates simple masks and that is obtained by a simple modification of *DoubleMasks*. We conducted experiments using two types of data sets. The first dataset is a real blacklist downloaded from the repository http://iplists.firehol.org/. The second dataset is a list of synthetically generated IP addresses.

1) Simulation setup: The real blacklist dataset contains more than 1.5 million IP addresses that are collected from different sources and combined together. We first transform this set of IPs into ranges. Then we compare the effects of a double mask representation w.r.t. a simple mask representation in reducing the size of our dataset. To generate double masks we rely on *DoubleMasks* algorithm and to generate simple masks we rely on a simple modification of *DoubleMasks* called *SimpleMasks*. The two programs were coded in Java language. The

Algorithm 3 DoubleMasks(a,b)

Input: a, b
Output: set of masks representing $[a, b]$
$c \leftarrow$ longest common prefix of a and b
$p \leftarrow c0, q \leftarrow c1$ ▷final result will be computed from siblings of c
$DM_p \leftarrow ComputeMasks(a, c, 0)$ ▷refer to Algo. 2
$DM_q \leftarrow ComputeMasks(b, c, 1)$ ▷refer to Algo. 2
if $DM_p = \{pr/|p|/0\}$ then
  if $DM_q = \{qs/|q|/0\}$ then
    return $\{cr/|c|/0\}$ ▷new SM generated
  else if $DM_q = \{c\{1\}^{w-|c|}/|q|/|s|\}$ then
    return $\{c1s/|q|-1/|s|+1\}$ ▷DM extended
  else if $DM_q = \{qs/|q|+1/0\}$ then
    return $\{c11t/|c|/2\}$ ▷new DM generated
  else
    return $DM_p \cup DM_q$
  end if
else if $DM_q = \{qs/|q|/0\}$ then
  if $DM_p = \{c\{0\}^{w-|c|}/|p|/|s|\}$ then
    return $\{c0s/|p|-1/|s|+1\}$ ▷DM extended
  else if $DM_p = \{ps/|p|+1/0\}$ then
    return $\{c00t/|c|/2\}$ ▷new DM generated
  else
    return $DM_p \cup DM_q$
  end if
else
  return $DM_p \cup DM_q$
end if

experiments are carried on a desktop computer with Intel core i7-7700 3.6-GHz CPU, 32 GB of RAM and running Windows 10 operating system.

We define the following metrics for analysing the performance of the two algorithms:

$$\text{Average Compression Ratio} = 1 - \frac{M}{n*S}$$

where

M is the number of masks generated in all iterations,
S is the number of IPs in the dataset,
n denotes the number of iterations.

To compute the average compression ratio, the number of iterations is set to 20. We use this metric to show that our algorithm can generate a more compact list of rules in comparison with *SimpleMasks* algorithm.

2) Results of real life IP blacklist: The blacklist IPs are aggregated into approximately 6000 ranges. In order to have a much larger ranges, the two algorithms will take as input all the ranges located between the set of ranges computed previously. The two programs take as input each range and compute a set of masks covering this range. Fig. 5 shows the distribution of range lengths used in this experiment. We observe, that several ranges have very large lengths, but our algorithm is not sensitive to range length since it only processes the two IP boundaries of a range.

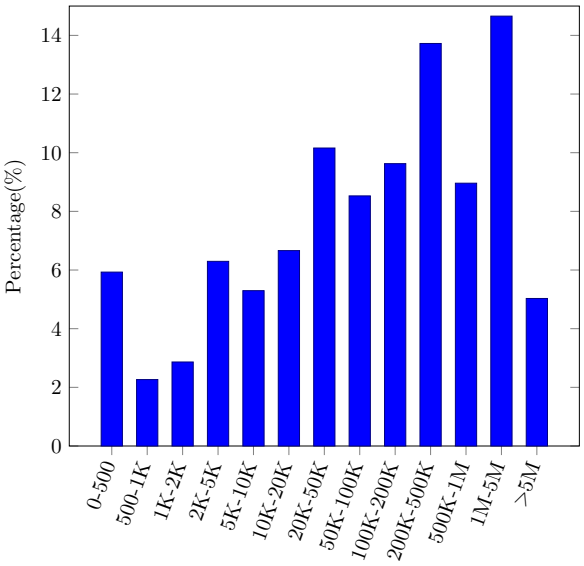

Fig. 5: Distribution of range lengths in the IP blacklist dataset.

Fig. 6 compares the number of masks generated by the two algorithms. By using double mask representation, we are able to reduce the number of masks by more than 11%. In total, 7% of generated masks are double masks (i.e. 3088 DM). As the number of ranges increases, we observe that *DoubleMasks* algorithm generates less masks than *SimpleMasks*. In this example, from the 6000 ranges only 15 are perfect ranges, and 13 are of the form $[1, 2^w - 1]$ or $[1, 2^w - 2]$. As discussed before, the real benefit of double masks to have a large compression ratio is obtained with these type of ranges. The limited number of this type of ranges in the blacklist dataset explains why the difference in the number of masks generated by the two algorithms is only 11%.

3) Synthetically generated dataset: In the second experiment, we conducted an evaluation over 6000 ranges computed from more than 1.5 millions IPs obtained in a synthetic way. Fig. 7 shows the difference between the total number of masks computed respectively by the two algorithms. In this scenario, we observe a large difference between simple and double mask techniques. The total number of generated simple masks is 29958. Using *DoubleMasks* algorithm, we are able to reduce this number by 74% (i.e. 7872 masks). The synthetic dataset used in Fig. 7 contains a higher number of ranges of the form $[1, 2^w - 1]$ which explains the difference between the obtained number of double and simple masks.

Fig. 8 shows the average compression ratio of the two algorithms while increasing the number of IPs. We observe, that *DoubleMasks* algorithm performs better than *SimpleMasks* with a difference of at least 10%.

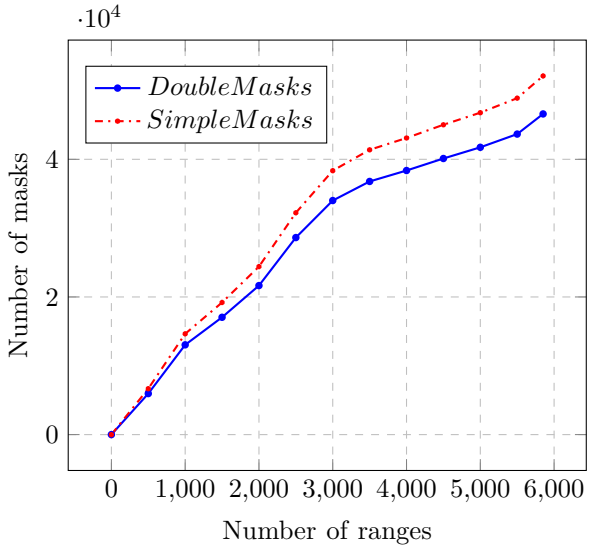

Fig. 6: Number of masks computed respectively by *DoubleMasks* and *SimpleMasks* algorithms using the IP blacklist dataset.

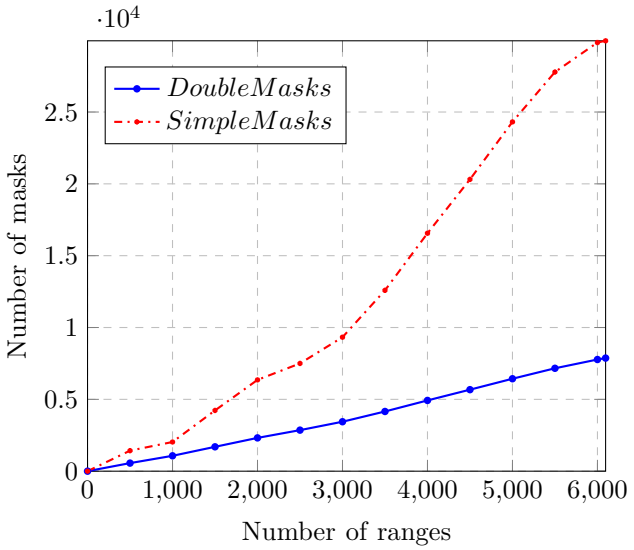

Fig. 7: Number of masks generated respectively by *DoubleMasks* and *SimpleMasks* algorithms using the synthetic dataset.

Fig. 9 shows the difference in compression ratio between *DoubleMasks* and *SimpleMasks* while modifying the length of IPs. We observe that *DoubleMasks* always performs better than *SimpleMasks* for each length value.

The compression ratio depends on each dataset and on the nature of IPs ranges. We use two types of datasets in order to demonstrate that this technique can reduce the number of rules by 79% and more in some cases and by 11% or less in others depending on the nature of IPs ranges. Since *DoubleMasks* algorithm generates a simple mask when no double mask can be generated,

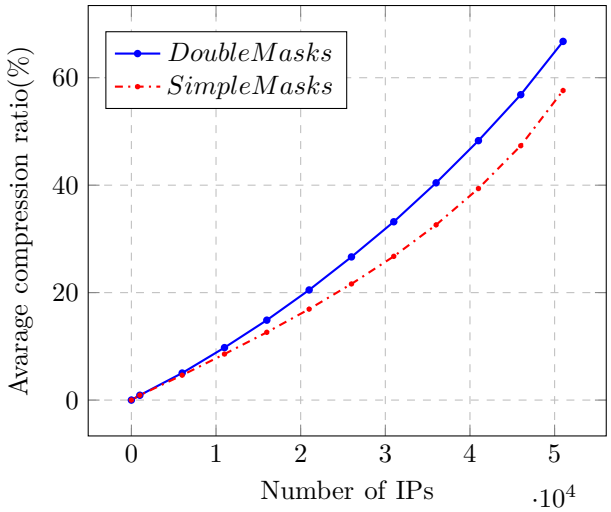

Fig. 8: Compression ratio of *DoubleMasks* and *SimpleMasks* using a synthetic dataset of range fields of length 16bits.

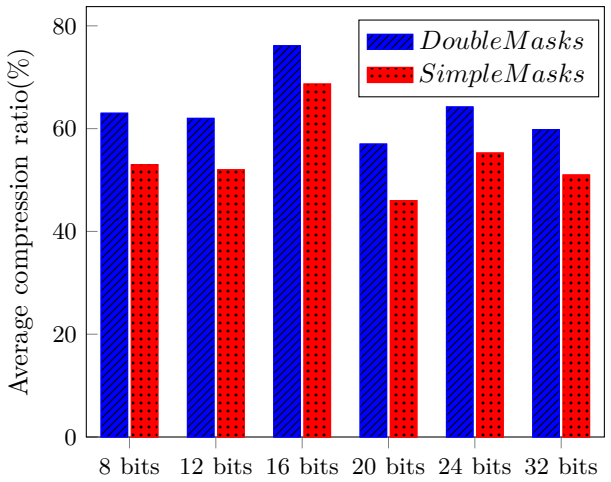

Fig. 9: Comparison of compressions ratio between *DoubleMasks* and *SimpleMasks* while varying the length of a field.

the total number of masks will be at most equal to the number of simple masks computed by *SimpleMasks*. This is why, according to our empirical simulations, *SimpleMasks* cannot generate a smaller set of masks than *DoubleMasks*.

VII. Hardware Implementation

A. Architecture

In this work, we implement and set-up the architecture shown in Figure 10, which consists of a controller, a switch and multiple hosts.

In our set-up, we use the OpenFlow-enabled zodiac FX switch that provides an inexpensive alternative to experiment SDN networks in hardware. The switch provides

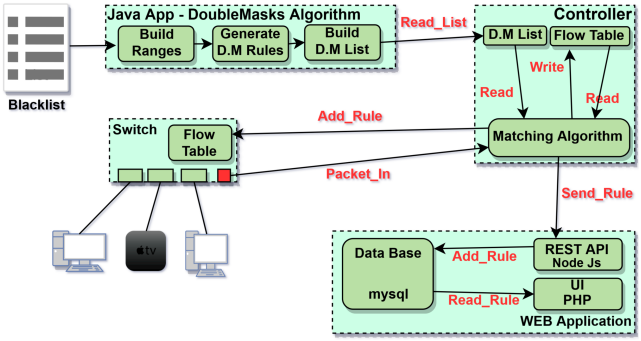

Fig. 10: Architecture and set-up of our SDN network.

four ports, one for the controller and the three others are to be used for hosts as shown in Figure 11.

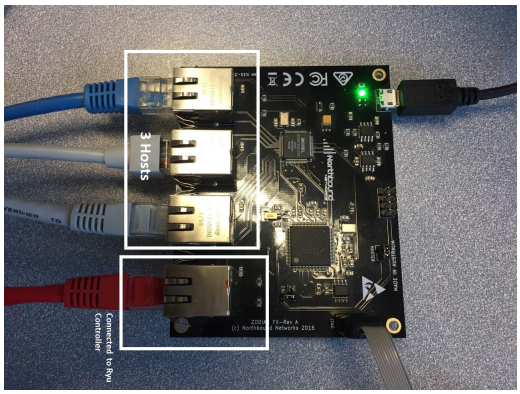

Fig. 11: The Zodiac Fx switch set-up.

The switch communicates using the Openflow protocol with an SDN controller. We used the Ryu SDN controller [28] that runs on a machine connected via Ethernet cable to the zodiac Fx switch. Ryu controller supports several protocols such as OpenFlow and it is developed in Python. The rules containing the double mask representation of IP matching fields are provided to the controller to install them in the switch. When the switch receives a packet it will perform a matching between the IP source and all the rules in the routing table, if the switch finds a match, the action of the rule that matches the IP will be performed.

B. Extending the control and data plane with the double-mask representation

We modified the control plane of our SDN system which the Ryu controller in order to integrate the double mask representation in the OpenFlow protocol match fields. In our modified implementation of Ryu, when the controller is provided with a rule containing a double mask it computes in the matching field the first and second masks and installs the rule in the flow table of the switch.

For the data plan component, we modified the OpenFlow implementation of the Zodiac Fx switch [29] to integrate the processing of rules containing the double mask representation. The code of this switch is open source and developed in C language. This code has also been modified in order to store a double mask in the forwarding table and to apply a matching between the IP source of a packet and all the rules in the switch. The Algorithm 4 has been integrated in the code of the switch to support the matching of a double-mask based OpenFlow rule and either the IP source or destination fields of a packet. If the value of S0 is zero, that means that the IP matches the network prefix ($netref$). If the value in S1 is different from zero that means that the IP is not included in the set of rejected IPs by $mask2$, in this case, the IP will match the rule in the switch.

Algorithm 4 Matching($netref, mask1, mask2, ip$)

```
1: Input: netref, mask1, mask2, ip
2: Output: accept or deny
3: AND1 ← mask1∧ netref
4: XOR1 ← AND1 ⊕ ip
5: S0 ← XOR1 ∧ mask1
6: if S0 = 0 then
7:                                  ▷ip match the prefix of the network
8:    OR ← mask1∨mask2
9:    AND2 ← OR ∧ip
10:   AND3 ← OR ∧ netref
11:   S1 ← AND2 ⊕ AND3
12:   if S1 # 0 then
13:                          ▷ip not included in IPs rejected by mask2
14:      return accept
15:   else
16:      return deny
17:   end if
18: else
19:   return deny
20: end if
```

The Algorithm 4 has been integrated in the code of the switch to support the matching of a double-mask based OpenFlow rule and either the IP source or destination fields of a packet. If the value of S0 is zero, that means that the IP matches the network prefix ($netref$). If the value in S1 is different from zero that means that the IP is not included in the set of rejected IPs by $mask2$, in this case, the IP will match the rule in the switch. When the controller matches a packet, the rule is sent to both the switch and the Web Application. The switch then adds the rule to the flow table order to perform the matching directly without the need of the controller. The rule is also added to the database in order to perform some statistics like specifying the importance of some rules based on the number of times a packet is being matched to a specific rule.

VIII. Related Works

Reducing the number of rules in a firewall is a very common problem that has been studied in multiple works. For instance, [14] proposes an approach to detect anomalies in firewall rules like generalization, shadowing and correlation and recommends actions for correcting those anomalies in order to reduce the number of rules and increase the performance of firewalls. In [6], a new compression scheme was presented to minimize the

number of policies in a firewall by removing redundant and shadowed rules. In [10], the authors present a new aggressive reduction algorithm by merging rules together using two-dimensional representation.

Since TCAM is the standard for rules storage and matching in packet classification for Openflow switches, multiple attempts to solve their problems was considered in [15]–[19]. To reduce the number of entries in TCAM, [9] proposes a new algorithm to remove redundant rules using a tree representation. On the other hand, [11] proposes a new compiler that aims to reduce the number of entries in switches and to speed up the packet classification process. In [20] a new systematic approach was introduced to minimize the prefix rules in TCAM. A mechanism called "Flow Table Reduction Scheme" has been introduced in [21] to minimize the number of flow entries in SDN. This paper focuses on reducing the number of entries by using a new representation for IP ranges, since reducing the number of entries can improve the power consumption of TCAM, while respecting the capacity constraint.

The number of prefixes needed to cover a range has also been studied extensively in the literature. In [22], the authors show that by using Gray encoding, the number or intervals needed is also $2w-4$. Despite having the same upper bound with double mask approach, our technique can be more efficient in some cases. For example, the range [6,14] mentioned in [22], need three entries to be represented using gray code but two using a double mask. The DNF (disjunctive normal form) has also been applied to compute the minimal Boolean expression for a range in linear time [23] and to prove the $2w-4$ upper bound. The works above admit only "accept" actions. Several works have also addressed the minimization of the number of entries with both "accept" and "deny" actions. In this case the upper bound can reach w entries [24], [25]. However the order of rules is very important in these approaches and rules management gets more complex. Our work is software-based, and relies only on accept rules, unlike [8], [24], [25]. We rely on a notation proposed in [12] that can reduce dramatically the number of entries in routing tables. In comparison, representing a w-bit range may need $2w-2$ prefixes [26]. For example $[1, 14]$ needs 6 entries but with the double-mask notation two entries are sufficient. This new notation has the same upper bound of $2w-4$ presented in other papers [22], [23], [27], but in some cases, the number can be reduced as shown before in our experimental results.

IX. Conclusion and future work

The double mask representation has been informally introduced in [12] to reduce the number of rules in firewalls, IDS's or routing tables in order to make the configuration, the management and deployment easier.

In this paper, we formally propose the first linear algorithm to compute a set of double masks covering a range of IPs. Note that our algorithm can be applied after or in combination with known redundancy removal techniques [6] in order to further reduce the number of entries in filtering rule tables. Then we conducted a series of experiments on real and synthetic dataset. According to our experiments, using the double mask representation allows one to reduce the number of rules needed to cover a set of ranges by more than 11% on a real blacklist (after removing the redundant rules) and more than 74% on synthetic data. The algorithm is not limited to IP ranges and it can be applied to port ranges too and to reduce the range expansions in TCAM. Finally, we demonstrate by adding support of double masks in SDN networks, that this new representation can be used instead of simple mask.

Our future work consists of computing double masks for union of ranges in order to achieve a higher level of optimization in routing tables. We also plan to design fast update strategies of generated double masks to handle rapid changes in filtering policies.

Acknowledgement

This work is supported by a CIFRE convention between the ANRT (National Association of Research and Technology) and the company NUMERYX Technologies.

References

[1] bgphelp, 2017 BGP Table Size Prediction and Potential Impact on Stability of Global Internet Infrastructure, 2017. [Online]. Available: http://bgphelp.com/2017/01/01/bgpsize/

[2] A. Hiromori, H. Yamaguchi, K. Yasumoto, T. Higashino, and K. Taniguchi, "Reducing the size of routing tables for large-scale network simulation," in Seventeenth Workshop on Parallel and Distributed Simulation, 2003. (PADS 2003). Proceedings., June 2003, pp. 115–122.

[3] W. Braun and M. Menth, "Wildcard compression of inter-domain routing tables for openflow-based software-defined networking," in 2014 Third European Workshop on Software Defined Networks, Sept 2014, pp. 25–30.

[4] C. R. Meiners, A. X. Liu, and E. Torng, "Bit weaving: A non-prefix approach to compressing packet classifiers in TCAMs," IEEE/ACM Transactions on Networking, vol. 20, no. 2, pp. 488–500, April 2012.

[5] Symantec, Internet Security Threat Report, April 2017. [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf

[6] A. X. Liu, E. Torng, and C. R. Meiners, "Firewall compressor: An algorithm for minimizing firewall policies," in IEEE INFOCOM 2008 - The 27th Conference on Computer Communications, April 2008, pp. 176–180.

[7] V. Fuller, T. Li, J. Yu, and K. Varadhan, "Classless inter-domain routing (CIDR): An address assignment and aggregation strategy," United States, 1993.

[8] N. B. Neji and A. Bouhoula, "Naf conversion: An efficient solution for the range matching problem in packet filters," in 2011 IEEE 12th International Conference on High Performance Switching and Routing, July 2011, pp. 24–29.

[9] Y. Sun and M. S. Kim, "Tree-based minimization of TCAM entries for packet classification," in 2010 7th IEEE Consumer Communications and Networking Conference, Jan 2010, pp. 1–5.

[10] M. Yoon, S. Chen, and Z. Zhang, "Reducing the size of rule set in a firewall," in 2007 IEEE International Conference on Communications, June 2007, pp. 1274–1279.

[11] S. Hommes, P. Valtchev, K. Blaiech, S. Hamadi, O. Cherkaoui, and R. State, "Optimising packet forwarding in multi-tenant networks using rule compilation," in 2017 IEEE 16th International Symposium on Network Computing and Applications (NCA), Oct 2017, pp. 1–9.
[12] A. Bouhoula and N. B. Neji, "Double-masked IP filter," Patent, 04 10, 2015. [Online]. Available: https://bases-brevets.inpi.fr/fr/document/FR3011705.html
[13] D. E. Taylor, "Survey and taxonomy of packet classification techniques," ACM Comput. Surv., vol. 37, no. 3, pp. 238–275, Sep. 2005. [Online]. Available: http://doi.acm.org/10.1145/1108956.1108958
[14] A. Bouhoula, Z. Trabelsi, E. Barka, and M. Anis Benelbahri, "Firewall filtering rules analysis for anomalies detection," IJSN, vol. 3, pp. 161–172, 01 2008.
[15] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small forwarding tables for fast routing lookups," SIGCOMM Comput. Commun. Rev., vol. 27, no. 4, pp. 3–14, Oct. 1997. [Online]. Available: http://doi.acm.org/10.1145/263109.263133
[16] H. Liu, "Efficient mapping of range classifier into ternary-cam," in Proceedings 10th Symposium on High Performance Interconnects, Aug 2002, pp. 95–100.
[17] Q. Dong, S. Banerjee, J. Wang, D. Agrawal, and A. Shukla, "Packet classifiers in ternary cams can be smaller," SIGMETRICS Perform. Eval. Rev., vol. 34, no. 1, pp. 311–322, Jun. 2006. [Online]. Available: http://doi.acm.org/10.1145/1140103.1140313
[18] E. Spitznagel, D. Taylor, and J. Turner, "Packet classification using extended TCAMs," in 11th IEEE International Conference on Network Protocols, 2003. Proceedings., Nov 2003, pp. 120–131.
[19] O. Rottenstreich, I. Keslassy, A. Hassidim, H. Kaplan, and E. Porat, "On finding an optimal TCAM encoding scheme for packet classification," in 2013 Proceedings IEEE INFOCOM, April 2013, pp. 2049–2057.
[20] C. R. Meiners, A. X. Liu, and E. Torng, "TCAM razor: A systematic approach towards minimizing packet classifiers in TCAMs," in 2007 IEEE International Conference on Network Protocols, Oct 2007, pp. 266–275.
[21] B. Leng, L. Huang, C. Qiao, H. Xu, and X. Wang, "Ftrs: A mechanism for reducing flow table entries in software defined networks," Computer Networks, vol. 122, pp. 1 – 15, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128617301470
[22] A. Bremler-Barr and D. Hendler, "Space-Efficient TCAM-Based Classification Using Gray Coding," IEEE Transactions on Computers, vol. 61, no. 1, pp. 18–30, Jan 2012.
[23] B. Schieber, D. Geist, and A. Zaks, "Computing the minimum DNF representation of boolean functions defined by intervals," Discrete Applied Mathematics, vol. 149, no. 1, pp. 154 – 173, 2005.
[24] R. Cohen and D. Raz, "Simple efficient TCAM based range classification," in 2010 Proceedings IEEE INFOCOM, March 2010, pp. 1–5.
[25] O. Rottenstreich, R. Cohen, D. Raz, and I. Keslassy, "Exact worst case TCAM rule expansion," IEEE Transactions on Computers, vol. 62, no. 6, pp. 1127–1140, June 2013.
[26] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," SIGCOMM Comput. Commun. Rev., vol. 28, no. 4, pp. 191–202, Oct. 1998. [Online]. Available: http://doi.acm.org/10.1145/285243.285282
[27] T. Sasao, "On the complexity of classification functions," in 38th International Symposium on Multiple Valued Logic (ismvl 2008), May 2008, pp. 57–63.
[28] Ryu OpenFlow controller. [Online]. Available: https://osrg.github.io/ryu/
[29] NorthboundNetworks, Zodiac Fx switch. [Online]. Available: https://github.com/NorthboundNetworks/ZodiacFX