Double Mask: An efficient rule encoding for Software Defined Networking

Ahmad Abboud, Abdelkader Lahmadi, Michael Rusinowitch, Miguel Couceiro, Adel Bouhoula, Mondher Ayadi

To cite this version:

Ahmad Abboud, Abdelkader Lahmadi, Michael Rusinowitch, Miguel Couceiro, Adel Bouhoula, et al.. Double Mask: An efficient rule encoding for Software Defined Networking. ICIN 2020 - 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops, Feb 2020, Paris, France. pp.186–193. hal-02547097

HAL Id: hal-02547097

https://hal.archives-ouvertes.fr/hal-02547097

Submitted on 19 Apr 2020

Double Mask: An efficient rule encoding for Software Defined Networking

Ahmad Abboud[†*], Abdelkader Lahmadi[*], Michael Rusinowitch[*], Miguel Couceiro[*]
Adel Bouhoula[‡†], 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, 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. 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 evaluate the performance of our double masks representation through an OpenFlow based implementation with an SDN testbed using real hardware. Our results show that our technique is capable of significantly reducing the matching time in the controller when compression ratios are higher than 15% leading to a faster response time, and a good balance between matching time and memory space in the switch.**

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.

On the other hand, attacks on Internet have reached a high level according to [1]. The number keeps increasing which in turn increases the size of blacklists and the number of rules in firewalls. The limited storage capacity [2] requires efficient management of that space.

To face the large number of hosts and routing tables, [3] developed Classless Inter-Domain Routing (CIDR) to replace the classful network architecture. 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 [4].

In this work, our main goal is to find a simple representation of filtering rules that enables more compact rule tables and thus easier 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*, 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 the alternative representations and therefore can prevent errors in network management operations. In this paper, we demonstrate the practicality of this representation over range rules. We provide an efficient algorithm (linear time) that is able to build the double mask representation of a set of range rules. And finally, we add support to double mask in OpenFlow and we evaluate the performance of the matching algorithm using a double mask instead of a simple mask. To summarize, our contributions are fourfold:

1) We formally define the double masks representation over range rules.
2) We design a linear algorithm to transform range rules into a double masks 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.
4) We implement the double masks representation and matching in OpenFlow implementation both on an SDN switch and a controller in order to evaluate its performance and impact.

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 and we present the strategy behind our linear algorithm for building double masks representations over range fields. In Section IV, we describe our experiments and the performance evaluation results of the proposed algorithm using real-life and synthetic datasets. In Section IV-B we evaluate the performance of our matching algorithm for the double mask by implementing the new representation into an SDN architecture. Related works are discussed in Section V. In Section VI 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 address
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)
$\lvert p\rvert$	length of p
$t(p)$	perfect binary subtree of t_w with root p
$l(p)$	set of leaves of $t(p)$
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 $ip_i = p_i \quad for\ i \in [k, k - i + 1]$. 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 $0 \leq a \leq b \leq 2^w - 1$). 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 $\leq w$ 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.

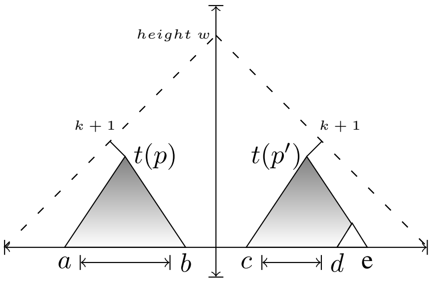


Fig. 1. Illustration of a perfect Binary Tree.

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

Now we will define the double mask 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* $ip_i = netpref_i \quad for\ i \in [1, \ldots, mask1]$ *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) address 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:

$$N1: \quad a_{k-1}...a_0a_{j-1}...a_00_{i-1}...0_0/k/j$$

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

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 the 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 :*

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

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 $DM_p = DM_{p0} \cup DM_{p1}$

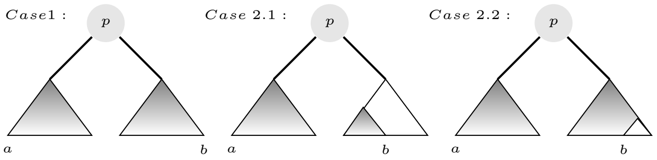


Fig. 2. Typical examples for *Cases* 1, 2.1 *and* 2.2

Algorithm 1 Generate-DMasks(a,b)

```
Input: a,b
Output: set of double masks representing [a,b]
return  $DM_\varepsilon$ where:
if $p$ is a leaf then
    if $p \notin [a, b]$ then
        return  $DM_p = \emptyset$
    else

        return  $DM_p = \{p/|p|/0\}$                                   ▷Case 0
    end if
end if
if $l(p0), l(p1) \subseteq [a, b]$ then

    return  $DM_p = \{p0^{w-|p|}/|p|/0\}$                             ▷Case 1
else
    if $l(pi) \subseteq [a, b]$ then
        if $DM_{p\bar{i}} = \{p\bar{i}dq/|p| + 2/0\}$ $(d \in \{0, 1\})$ then

            return  $DM_p = \{p\bar{i}\bar{d}q/|p|/2\}$                  ▷Case 2.1
        else
            if $DM_{p\bar{i}} = \{p\bar{i}q/|p| + 1/m\}$   $(m > 0)$ then

                return  $DM_p = \{p\bar{i}q/|p|/m + 1\}$               ▷Case 2.2
            end if
        end if
    end if
end if

return  $DM_p = DM_{p0} \cup DM_{p1}$                                  ▷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.

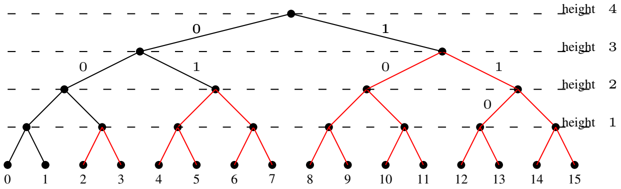


Fig. 3. Example : Let $[a, b] = [2, 15]$. The algorithm start from the button. $[2, 2]$ is a leaf and $\in [a, b]$. According to Case 0, $DM_{0010} = \{0010/1/0\}$. Same for each leaf in $[a, b]$. At height 1, if $l(p0), l(p1) \subseteq [a, b]$, the algorithm return $DM_p = \{p0^{w-|p|}/|p|/0\}$. For $[2, 3]$, $DM_{001} = \{0010/3/0\}$ according to Case 1 since $l(p0) = 2$ and $l(p1) = 3$. 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* $[12, 15]$, $l(p0)$, $l(p1) \subseteq [a, b]$. For example, in $[4, 7]$, $l(010)$, $l(011) \subseteq [2, 15]$, the algorithm return $DM_{01} = \{p0^{w-|p|}/|p|/0\} = \{0100/2/0\}$ according to Case 1. 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 $DM = \{0000/0/3\}$.

By performing a case analysis, we can show the following results:

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

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

From these two propositions, we can easily see that the $2w - 4$ bound is tight.

A. Deriving a linear time algorithm

We introduce an efficient variant of our algorithm, named *DoubleMask*, to compute a set of masks covering a range $[a, b]$ for positive integers a and b.

Let $bin_k(a)$ and $bin_k(b)$ be the binary representations of a and b, respectively using k bits. The key idea is the following. Let c be the longest common prefix of $bin_k(a)$ and $bin_k(b)$. Then $bin_k(a) = c0a'$ and $bin_k(b) = c1b'$. The algorithm computes the set of masks for $[a, b]$ in a bottom up way, starting from the two nodes $bin_w(a)$ and $bin_w(b)$. When reaching node c, the sets of computed masks for $[c0a', c01 \cdots 1]$ and for $[c10 \cdots 0, c1b']$ are combined and the algorithm stops. This algorithm is linear in k where k is the number of bits to represent an IP address.

IV. Evaluation

We evaluate our double mask representation through simulation IV-A and an implementation of a proof of concept with openFlow in a physical SDN testbed. Our evaluation is focused on showing the compression gain by using double masks as well as its practical capability to reduce matching time both in an SDN switch and a controller.

A. Results from Simulation

We evaluate the performance of the Algorithm *DoubleMask* and we compare it with the algorithm that only generates simple masks and that is obtained by a simple modification of *DoubleMask*. We conducted experiments using two types of data sets. The first dataset is a real IP 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 *DoubleMask* algorithm and to generate simple masks we rely on a simple modification of *DoubleMask* called *SimpleMask.*

The two programs were coded in Java language. The 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 *SimpleMask* algorithm.

2) Results from 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. 4 compares the number of masks generated by the two algorithms. By using double masks 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 *DoubleMask* algorithm generates less masks than *SimpleMask*. 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 at 11%.

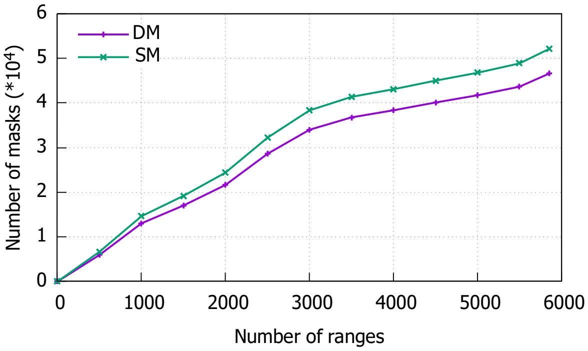


Fig. 4. Number of masks generated respectively by *DoubleMask* and *SimpleMask* algorithms using the IP blacklist dataset.

3) Results from 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. 5 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 *DoubleMask* algorithm, we are able to reduce this number by 74% (i.e. 7872 masks). The synthetic dataset used in Fig. 5 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. 6 shows the average compression ratio of the two algorithms while increasing the number of IPs. We observe, that *DoubleMask* algorithm performs better than *SimpleMask* with a difference of at least 10%.

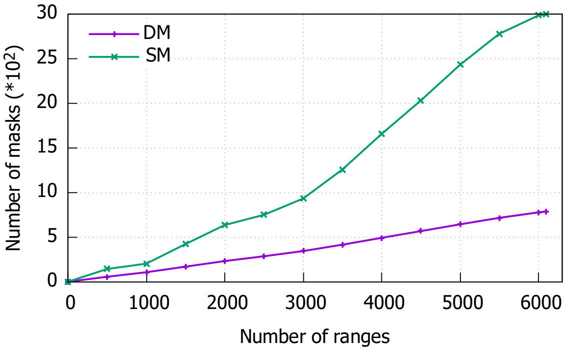


Fig. 5. Number of masks generated respectively by *DoubleMask* and *SimpleMask* algorithms using the synthetic dataset.

Fig. 7 shows the difference in compression ratio between *DoubleMask* and *SimpleMask* while modifying the length of IPs. We observe that *DoubleMask* always performs better than *SimpleMask* for each length value.

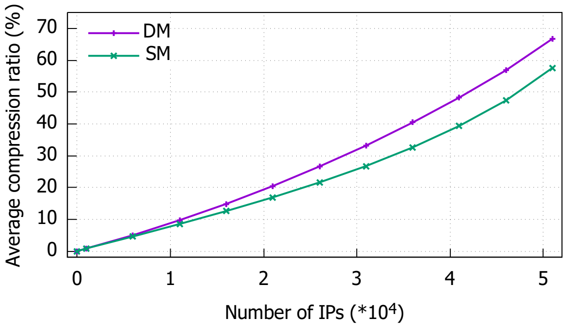


Fig. 6. Compression ratio of *DoubleMask* and *SimpleMask* using a synthetic dataset of range fields of length 16bits.

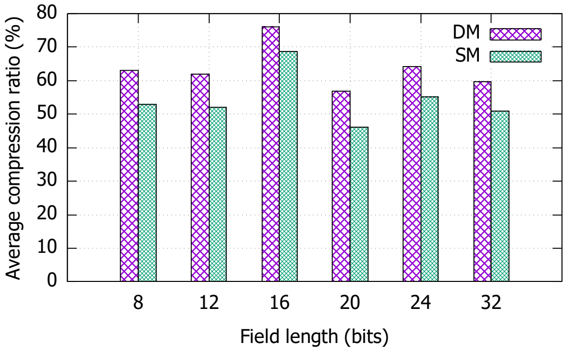


Fig. 7. Comparison of compressions ratio between *DoubleMask* and *SimpleMask* while varying the length of a field.

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 *DoubleMask* algorithm generates a simple mask when no double mask can be generated, the total number of masks will be at most equal to the number of simple masks computed by *SimpleMask*. This is why, according to our empirical simulations, *SimpleMask* cannot generate a smaller set of masks than *DoubleMask*.

B. *Experimental Evaluation*

We implement the double masks representation and its respective matching algorithm using an OpenFlow switch and a SDN controller. We evaluate and compared the performance of our matching algorithm with simple masks representation considered as a baseline.

1) Experimental setup and parameters: In this setup, we use the physical SDN testbed shown in Figure 8. The testbed contains a Zodiac FX switch [5] connected to a RYU controller [6] and three hosts. The code in the controller and the Switch has been modified with double masks representation for IP matching fields. A matching function has also been added in both of them in order to match a received packet with the set of rules in the routing table of the switch or with a list of rules

in the controller.
In this scenario, a host machine (host 3) connected to the switch takes a list of IP addresses then sends packets to those destinations. Another host (host 1) tries to send a ping message to host 2. The ping message will match all rules in the routing table of the switch before forwarding the packet to host 2. If no match is found, the message will be sent to the controller who matches the message with the list of rules then send back the action needed for the specific packet to the switch. The round trip time (RTT) from host 1 to host 2 is recorded.

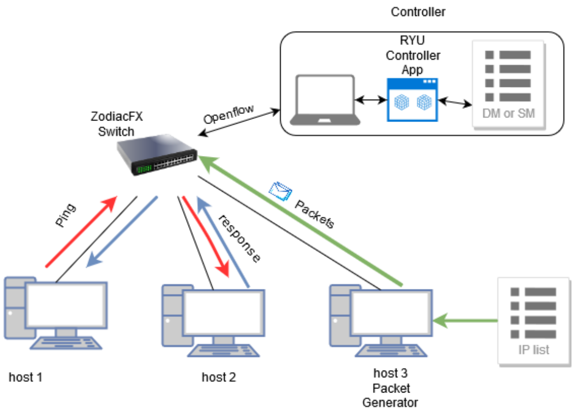


Fig. 8. The experimental physical SDN testbed.

a) Rules in the controller: The code of the RYU controller has been modified in order to integrate the double masks representation in the OpenFlow protocol match fields. The controller takes a set of rules (simple or double masks) computed from a blacklist using our transformation algorithm. The controller matches the header of the OpenFlow *PACKET_IN* messages with the set of rules in order to block or not the traffic to certain destinations.

b) Rules in Zodiac FX switch: In our set-up, we use the OpenFlow-enabled Zodiac FX switch that provides an inexpensive alternative to experiment SDN networks in hardware. The OpenFlow implementation of the Zodiac FX switch has been modified to integrate the processing of rules containing the double masks representation. This code has also been modified in order to apply a matching between the IP source of a packet and all the rules in the switch. The routing table of our Zodiac FX switch can store a maximum number of 448 rules using simple or double masks. The timeout for each rule is set to 30 seconds after that the rule will be automatically removed from the table. The switch has two matching algorithms. The first algorithm is the standard algorithm used to match a received packet with a simple mask rule in the routing table and the second one is used for the double mask rules.
Algorithm 2 implemented in the switch and the controller shows the matching process between the source IP and the rule in the flow table. If the value of $S0$ is zero that means the IP source matches the network IP. If the value of $S1$ is different from zero that means the IP source is not included in the set of rejected IPs by $mask2$, in this case, the IP source will match the rule.

Algorithm 2 Matching$(netref, mask1, mask2, ip_source)$

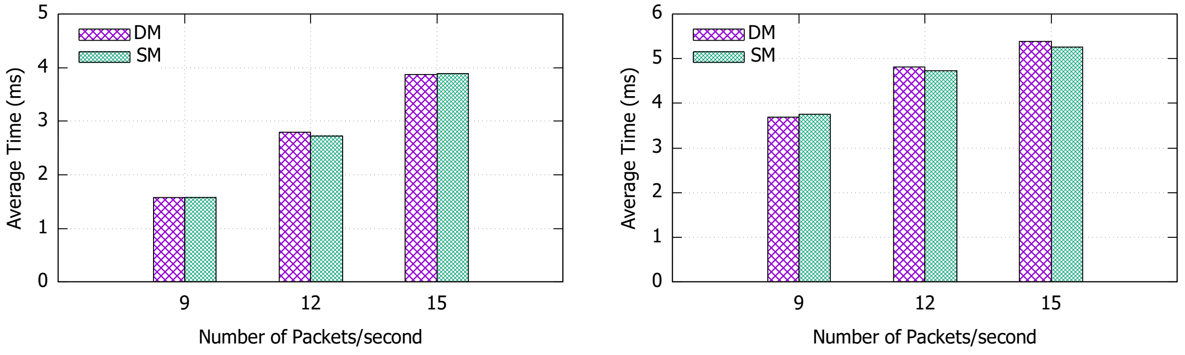
```
Input: $netref, mask1, mask2, ip\_source$
Output: $accept\ or\ deny$
$AND1 \leftarrow mask1 \wedge netref$
$XOR1 \leftarrow AND1 \oplus ip\_source$
$S0 \leftarrow XOR1 \wedge mask1$
if $S0 = 0$ then
                              ▷$ip\_source\ match\ the\ ip\ of\ the\ network$
    $OR \leftarrow mask1 \vee mask2$
    $AND2 \leftarrow OR \wedge ip\_source$
    $AND3 \leftarrow OR \wedge netref$
    $S1 \leftarrow AND2 \oplus AND3$
    if $S1 \# 0$ then
                    ▷$ip\_source\ not\ included\ in\ IPs\ rejected\ by\ mask2$
        return $accept$
    else
        return $deny$
    end if
else
    return $deny$
end if
```

c) Packet generator: The packet generator takes a list of IP addresses and then sends packets to each address in the list. We set the packet rate for each experiment to be 9, 12 or 15 packets/second. The packets are being sent to different destinations based on multiple datasets. The number of packets per second is chosen so that the time needed for the saturation of the switch routing table is around 10 min. If the number of packets is too small the table will never reach the maximum number of 448 rules, and if it is too large the table will max-out quickly. We repeat each experiment 6 times on 6 different datasets then we compute the average matching time. The same datasets are used to generate the different packets in order to match the sets of simple and double masks rules.

2) Matching time in the switch: In these experiments, the average matching time is computed after using a simple or a double masks list. Our goal is to study the impact of the compression ratio on the matching time in the switch. We will use two sets of rules the first one with a compression ratio of 5% and the second one with 30%. Figure 9 shows the difference in the average time between simple and double masks while changing the number of the packets generated at host 3. As shown in the two subfigures, the average matching time with simple and double masks are very close. Matching with a double mask is more costly than with a simple mask. However, this increase in time is compensated by a smaller number of rules in the routing table while using a double mask.

In order to see the real impact of the compression ratio on the global matching time, we will test the response time on the controller using the two matching functions for simple and double masks with multiple sets of rules with a different compression ratio.

3) Matching time in the controller: In a second experiment, we compare the matching time between a list of simple or double masks in the controller side. Our experiment uses 7 IP blacklists. Each blacklist generates two lists of rules one with only simple masks and the other with both simple and double masks. The compression ratio for the different sets of



(a) 5% compression ratio

(b) 30% compression ratio

Fig. 9. Average matching time measured in the switch with simple (SM) or double masks (DM) using : (a) 5% compression ratio or (b) 30% compression ratio.

lists varies between 0.5% and 83%. The goal here is to test the effect of the compression ratio on the response time of the controller. A set of 300K IPs is being used in order to match each IP address with each rule for the different sets. We use this number of IPs to simulate heavy traffic in order to show the gain in response time. First we compute the response time of the controller using only simple mask rules. Then we compute the response time using the same sets of IPs on a second set of rules that uses simple and double masks.

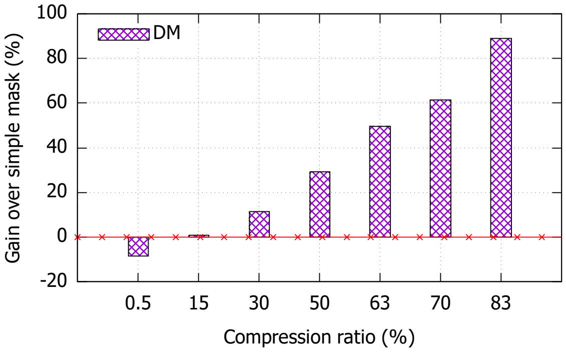


Fig. 10. Gain in the response time in the controller side when using double masks filters.

As shown in Figure 10, when the compression ratio is at 0.5% it is better to use a simple mask over a double mask since the gain in space is small in comparison with what we lose by using the matching function for the double mask that takes more time than the matching time for simple mask. At 15% we can see that the time needed for matching double or simple mask is similar. When the compression ratio is higher than 15%, we obtain substantial gain in response time by using double masks.

From a 30% compression ratio, the results show a gain in controller's matching time. However, at the same ratio, the gain in space is obtained in both: the switch and the controller.

V. Related Works

Reducing the number of rules in a firewall is a very common problem that has been studied in multiple works. For instance, [7] 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 [2], a new compression scheme was presented to minimize the number of policies in a firewall by removing redundant and shadowed rules. In [8], 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 [9], [10], [11], [12], [13]. To reduce the number of entries in TCAM, [14] proposes a new algorithm to remove redundant rules using a tree representation. On the other hand, [15] proposes a new compiler that aims to reduce the number of entries in switches and to speed up the packet classification process. In [16] a new systematic approach was introduced to minimize the prefix rules in TCAM. A mechanism called "Flow Table Reduction Scheme" has been introduced in [17] 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 [18], the authors show that by using Gray encoding, the number or intervals needed is also $2w - 4$. Despite having the same upper bound with the double mask approach, our technique can be more efficient in some cases. For example, the range [6,14] mentioned in [18], 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 [19] 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 [20], [21]. 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 [20], [21], [4]. Our notation can reduce dramatically the number of entries in routing tables. In comparison, representing a w-bit range may need $2w-2$ prefixes [22]. 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 [19], [18], but in some cases, the number can be reduced as shown before in our experimental results.

VI. Conclusion and future work

The double masks is a new representation used 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 [2] in order to further reduce the number of entries in filtering rule tables. Then we conducted a series of experiments on real and synthetic datasets. 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. We also evaluate the effectiveness of double masks using an OpenFlow based implementation and evaluate its matching time using a physical SDN testbed. Although the similar matching time in the switch between the simple and double masks representations, the storage space of rules is reduced. In the controller side, with compression ratios higher than 15% we observe a substantial gain in the matching and response times. 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] Symantec, *Internet Security Threat Report*, April 2017. [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf

[2] 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.

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

[4] 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.

[5] NorthboundNetworks, *Zodiac Fx switch*. [Online]. Available: https://github.com/NorthboundNetworks/ZodiacFX

[6] *Ryu OpenFlow controller*. [Online]. Available: https://osrg.github.io/ryu/

[7] 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.

[8] 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.

[9] 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.

[10] H. Liu, "Efficient mapping of range classifier into ternary-cam," in *Proceedings 10th Symposium on High Performance Interconnects*, Aug 2002, pp. 95–100.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] R. Cohen and D. Raz, "Simple efficient TCAM based range classification," in *2010 Proceedings IEEE INFOCOM*, March 2010, pp. 1–5.

[21] 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.

[22] 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.