A Formal Model and Technique to Redistribute the Packet Filtering Load in Multiple Firewall Networks

Luca Durante, Lucia Seno, *Member, IEEE*, and Adriano Valenzano, *Senior Member, IEEE*

***Abstract*— The dynamic redistribution of filtering rules between firewalls, which are located in the same network, is a technical solution that can cope with temporary changes in the traffic load processed by the firewalls themselves. This paper presents a novel formal model for networks including multiple cascaded firewalls, that can be leveraged to enable the transfer of a set of rules from a firewall to its downstream neighbors when the changes in the input traffic profile suggest to do so. With respect to other solutions appeared in the literature a formal approach, besides providing unambiguous specifications and mathematical proofs of correctness, also enables the computation of theoretical bounds for the expected performance before the proposed scheme is actually deployed in the target network. The underlying mechanism, on which our approach is based, is the reduction of the average number of rules checked per packet in order to increase the packet processing rate. Our network model takes into account both the system topology and firewall characteristics. A suitable transformation algorithm is then introduced, which is able to preserve the security integrity of the network while moving rules between cascaded firewalls and allowing tangible performance improvements in terms of packets processing rate for a given traffic profile. Correctness of the proposed solution has been formally proven and validated by means of simulation. Performance figures have also been obtained by running the proposed algorithm in a laboratory experimental test-bed.**

***Index Terms*— network security, firewall, rule distribution, formal methods, industrial communication networks.**

I. Introduction

TODAY, protection and effective management of digital communication networks (DCNs) in all application areas are recognized key aspects, which are gaining increasing attention even in domains that were not particularly sensitive to security issues till few years ago. Typical examples are networked automation systems, factory networks and distributed critical infrastructures [1], [2], which have become more and more prone to cyber-attacks since they started to migrate from proprietary communication technologies to more open, Internet-based solutions, and where the awareness for improved security guarantees is constantly rising [3], [4].

Filtering devices, in general, and firewalls (FWs), in particular, are popular hardware (h/w) and software (s/w) elements that are widely employed in digital networks a) to block unwanted traffic (erroneously) forwarded to any destination, b) to prevent malicious messages from reaching their targets and c) to deploy countermeasures against attacks originating from the cyberspace [5].

As networks grow in complexity, also because of powerfully emerging paradigms such as edge and fog computing [6], [7], Internet of Things (IoT) [8], [9], Industrial IoT (IIoT) [10], [11] and Industry 4.0 [12], [13], the number of firewalls they include also increases and hierarchies of (cascaded) h/w and/or s/w FWs can be found more and more frequently in real systems.

Unfortunately, FWs can also become performance bottlenecks and even preferential targets for threats such as denial-of-service (DoS) attacks. By its nature, FW filtering mainly consists of checking each incoming packet against a sequence of rules in order to decide whether the packet has to be either blocked or forwarded. Thus the filtering performance heavily depends on the (average) number of rules that are checked before making the decision. If the FW input load becomes too high, as in temporary traffic peaks or in DoS attacks, the latency experienced in packet processing can become too large and even cause packet losses.

To mitigate this problem, different solutions can be adopted, which either operate within a single FW, i.e., *intra-firewall* techniques, or rely on suitable network-level approaches involving multiple FWs, i.e., *inter-firewall* approaches.

Intra-firewall contributions appeared in the literature include, for instance, (optimal) rule ordering [14]–[18], firewall compression [19], [20] and rule analysis [21], [22]. Inter-firewall alternatives are frequently based on special architectures such as parallel firewalls [23]–[25]; another approach leverages the transfer of filtering rules among firewalls located in the same network [26], [27].

In this paper we propose a formal approach and an algorithm based on the redistribution of rules between cascaded firewalls in order to mitigate the performance loss in terms of packet processing rate experienced by a FW, when it becomes overloaded with the flow of packets it has to filter. With respect to other techniques appeared in the literature our solution does not require any change neither to the original routing of packets nor to their formats and fields. On the one hand, this aspect is particularly important in those situations, such as many factory, automation and process control networks, where the underlying communication infrastructure is not flexible enough to support dynamic re-configurations or the network is not able to tolerate freezes for offline modifications without

Manuscript received June 11, 2020; revised October 15, 2020 and December 20, 2020; accepted January 21, 2021. Date of publication February 8, 2021; date of current version March 1, 2021. This work was supported in part by the EU H2020-SU-ICT-03-2018 Project No. 830929 CyberSec4Europe (cybersec4europe.eu). The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Xiaodong Lin. *(Corresponding author: Adriano Valenzano.)*

The authors are with the National Research Council of Italy, CNR-IEIIT, 10129 Torino, Italy (e-mail: adriano.valenzano@ieiit.cnr.it).

Digital Object Identifier 10.1109/TIFS.2021.3057552

compromising the effectiveness of the control system. For the same reasons, neither the introduction of new h/w FWs nor the dynamic instantiation of virtual FWs is feasible in these scenarios, as the system must often work 24/7 and cannot be stopped or reconfigured on the fly. Thus deployed FWs can become bottlenecks in heavy traffic load conditions [3], [28]. On the other hand, the technique proposed in this paper does not rely on the availability of special h/w. For instance, advanced FWs make use of content-addressable memories (CAMs) or even ternary CAMs (TCAMs) to speed-up their operations, but they are rarely found in industrial devices which are, by contrast, quite simple and equipped with relatively low-power computing resources, as their design focuses on (mechanical) robustness over performance. With such a kind of constraints a software redistribution of the filtering load enables a better use of the existing h/w without affecting negatively the system functional requirements. In our case, the penalty to be paid to achieve improvements to the load filtering capability of the target FW is an increase in the communication bandwidth usage for links connecting the FW itself to its downstream neighbors.

The main contributions of this work are therefore the following:

1) A network formal model, which is able to deal with multiple cascaded firewalls and can be adopted to describe most systems that are found in real applications.
2) A RulE DIstribution ALgorithm (REDIAL) to move rules from a given FW to firewalls downstream and decrease the average number of rules checked per packet, in order to reduce the packet processing activity by the FW of interest. Correctness of the proposed technique is also formally proven and verified by simulation.
3) A simple model which can help in predicting the performance improvement achievable with the FW transformation before actually deploying the proposed technique in the target network. Lower and upper bounds are provided for the obtainable gain in terms of packet processing rate by the firewall.
4) Performance results obtained by simulating our solution in a number of conditions that confirm its advantages and a good accordance with the theoretical model.
5) Some experimental performance figures obtained by running the proposed solution in a simple but realistic laboratory test-bed.

Our approach assumes the availability of some kind of orchestrator/supervisor, which has overall visibility of the security settings for the whole network, and is able to coordinate and manage the filtering actions of multiple firewalls. The reader should be warned that this requirement, which is also found in proposals by other authors, can introduce some implementation cost. However, this kind of support and products are not uncommon, as they have been made available since quite a long time in proprietary technologies, i.e., [29], [30], though they often involve the adoption of s/w and h/w devices from the same manufacturer(s). In addition, today well-assessed paradigms such as software-defined networking (SDN) [31]–[33] and network function virtualization (NFV) [34] enable the design and deployment of inter-firewall solutions, by providing the needed flexibility and control/data decoupling also in networks adopting open and heterogeneous devices.

In general, moving rules between different FWs could be difficult because of the existence of heterogeneous administration domains and authorities. In the application areas considered in this paper, however, this aspect is rarely an issue as industrial/enterprise networks are often under control of few coordinated entities (i.e., the information technology (IT) and operation technology (OT) departments) belonging to the same organization.

The remaining part of the paper is organized as follows: Sect. II introduces the adopted notation and our model, whereas Sect. III presents the proposed approach in details and, in particular, the $REDIAL$ algorithm together with the formal proof that preserves the security integrity of the network. Sect. IV deals with the computation of performance bounds for our solution and shows the results obtained by simulation. Experimental performance figures are also introduced in this section. Sect. V discusses some related works appeared in the literature, pointing out similarities and differences with respect to our solution and, finally, Sect. VI draws some conclusions.

II. Modeling of Multiple Firewall Architectures

A. Streamed Topology

The model we consider is able to formally describe real architectures and types of topology, while easing and making the presentation of the proposed technique effective. In particular, we are interested in networks which include three distinct classes of devices:

- *Filtering Devices (i.e., firewalls):* these elements can be modeled with one input and one output port, and are able to forward to their output (some subset of) packets received from their input. The filtering function is based on a suitable set of rules.
- *Routing Devices (i.e., switches and routers):* these elements are generally equipped with several input/output ports, and are able to properly forward packets from an input to one or more outputs on the paths towards their intended destinations.
- *End Devices (i.e., personal computers, servers, special purpose computing nodes):* these elements are the origin(s) and/or destination(s) of packets. They communicate through an input/output network interface to receive and send data.

  For the purpose of modeling, a whole *subnetwork* can be encapsulated into a suitable end device, if its building blocks and connections are not of interest, thus masking the subnetwork internal topology and unnecessary details. Similarly, real devices can be modeled by composing one ore more logical elements (i.e. a personal computer equipped with two or more network interfaces and running a software firewall can be modeled by combining one filtering element plus one routing and end device pair). This choice enables smoothed transformations of

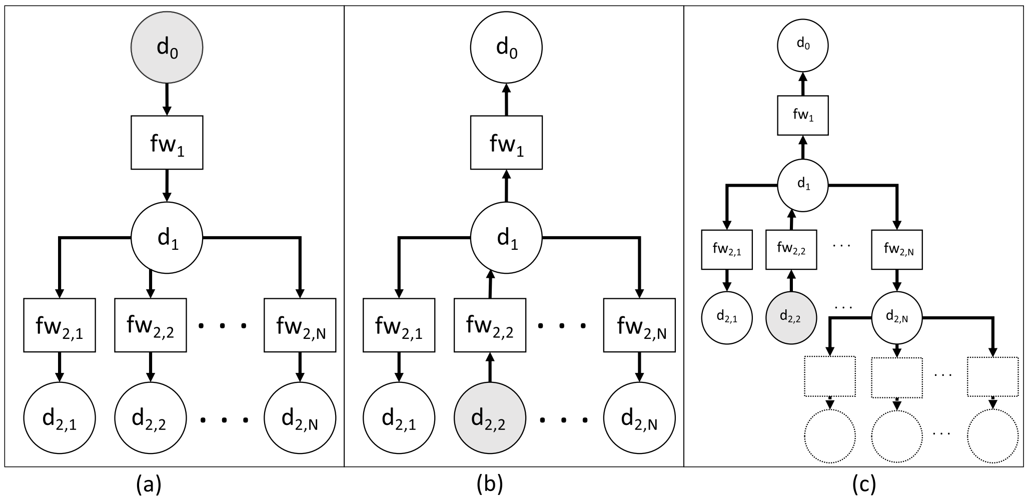

Fig. 1. Streamed topologies.

detailed and fine-grained architectures into simpler systems where one or more subnetworks are shrunk into a single device, and vice versa.

In a modeled network a logical firewall (possibly configured without any filtering rule) is always placed between any pair of not filtering devices and vice versa.

Thanks to the assumptions described above we have that:

- A direct (simple) link between any two devices can be modeled as a trivial firewall which forwards all the incoming packets.
- A direct link can also be seen as a trivial routing device.
- An end device can consist of a *shrunk subnetwork*, which can be either expanded or compressed recursively as necessary.

Flows are other main elements in our model. They take into account any stream of packets originating from an end device and flowing through links and intermediate elements to one or more destination devices. The class of systems we consider has a tree-shaped structure, where each source-to-destination path is rooted in the flow-originating device. In general, this holds for every node in the tree, and even leaf nodes can be seen as sub-trees, that can be recursively expanded/compressed as necessary. For this reason, a set of IP addresses can be associated with any end device.

Without any loss of generality, a network model can look like the one sketched in Fig. 1a: we call this structure *reference streamed topology* (RST). It is worth noting that different RSTs are possible for the same physical system, depending on the flow(s) being considered. This is shown in Fig. 1b, where the RST of interest is rooted in $d_{2,2}$, and can be expanded as in Fig. 1c by replacing $d_{2,N}$ with its shrunk subnetwork. In all cases the flow-originating device is grayed for better readability.

In the following, to keep the notation simple, RST device names are chosen so as to remind the sets of associated IP addresses. In Fig. 1a this implies that:

1) The IP address set d_0 for the root of the tree is known a priori.
2) The set $d_{2,i}$ for each leaf i is known a priori.
3) All sets for both the root and leaves are disjoint (not overlapped) and not interleaved.
4) The set for each routing device is the union of sets for each subtree rooted in it, e.g., $d_1 = \bigcup_{i=1}^{N} d_{2,i}$. In general, this set can be computed as the union of all sets in each subtree through a postorder visit which returns the set for each reached node and firewalls are treated as simple links.

B. Firewalls

We specifically consider IP-layer firewalls, whose filtering fields are source and destination IP addresses, source and destination port addresses and protocol number. However, the following definitions are general enough and might be adopted to describe firewalls operating at any network level.

Definition 1: A d-tuple of ranges $F = (f_1, \dots, f_d)$ is a tuple of d finite ranges of non-negative integers, $f_i \subset \mathbb{N}_0 \; \forall i \in [1, d]$.

As an example, we can define F to describe all the filtering fields taken into account by a firewall and, consequently, the corresponding fields in network packets: $F = ([0, 2^{32} - 1], [0, 2^{32} - 1], [0, 2^{16} - 1], [0, 2^{16} - 1], [0, 2^8 - 1])$, respectively representing source and destination IPv4 addresses, source and destination port addresses, and protocol number.

Definition 2: Given a d-tuple of ranges $F = (f_1, \dots, f_d)$, let's define a packet P over F as:

$$P = (p_1, \dots, p_d) \text{ where each } p_i \in f_i \,.$$

Definition 3: Given a d-tuple of ranges $F = (f_1, \dots, f_d)$, $\mathcal{P}$ is the set of all possible packets over F, i.e.

$$\mathcal{P} = \{P = (p_1, \dots, p_d) \text{ where each } p_i \in f_i\} \,,$$

and its cardinality is $|\mathcal{P}| = \prod_i |f_i|$, where $|f_i|$ stands for the cardinality of f_i, i.e., the width *of its corresponding range.*

Definition 4: Given a d-tuple of ranges $F = (f_1, \dots, f_d)$, let's define a condition *C over F as:*

$$C = (c_1, \dots, c_d) \text{ where each } c_i \subseteq f_i \,.$$

Definition 5: A condition C over $\mathcal{P}$ defines $\mathcal{P}_C \subseteq \mathcal{P}$ where

$$\mathcal{P}_C = \{(p_1, \dots, p_d) \in \mathcal{P} \;\text{ where each } p_i \in c_i\} \,,$$

and its cardinality is $|\mathcal{P}_C| = \prod_i |c_i| \leq |\mathcal{P}|$.

```
packet $f$( firewall $fw$, packet $P$ ) {
  if( $P = '-'$ ) return $'-'$;
  for( int $i \leftarrow 1$; $i \leq |fw|$; ) {
    if( $P$ matches $fw.r_i$ ) {
     if( $fw.r_i.action =$ goto k ) $i \leftarrow k$;
     else if( $fw.r_i.action =$ allow ) return $P$;
     else return $'-'$; /* $fw.r_i.action =$ deny */
    }
    else $i++$;
  }
}
```

Fig. 2. Firewall filtering operations.

Definition 6: A firewall fw is a finite sequence of rules

$$fw = (r_1, r_2, \ldots, r_{|fw|})\,,$$

where the number of rules $|fw|$ is called firewall cardinality. Each rule is defined as

$$r = (C, action)\,,$$

and $action$ varies in the finite set of all possible firewall actions Λ.

Definition 7: Given a packet P and a rule $r = (C, action)$, where both P and C share the same d-tuple of ranges F, we say that P matches r iff $P \in \mathcal{P}_C$.

The packet filtering activity performed by a firewall consists of searching a matching rule for any incoming packet, and in applying the corresponding action to the packet itself. This is schematically shown in Fig. 2. Since a packet could match, in principle, more than one rule in the firewall set and the matched rules might correspond to conflicting actions, a strategy must be defined to unambiguously select the rule and action to be performed. Typically, firewall rules are checked sequentially and, as soon as a match is found for the incoming packet, the corresponding action is executed (first matching strategy, FMS). Although other strategies could also be adopted, FMS is the most common in many real devices, therefore we will assume the use of FMS in the following.

Similarly, we assume that the set of all possible firewall actions is $\Lambda = \{allow,\ deny,\ goto\ n\}$, that is a firewall can either forward ($allow$) or discard ($deny$) any received packet, unless further analysis is needed and the process continues by checking the rule having index $n \in [1, |fw|]$. As a matter of fact, we are not interested in more subtle differences between possible actions such as $drop$, $reject$ and/or log, as we focus only on the firewall block/forward filtering activity.

A firewall fw is said to be *complete* if the analysis results in any packet $P \in \mathcal{P}$ matching at least one rule in fw, whose associated action is either $allow$ or $deny$. In real systems, a firewall can have multiple sub-sequences of rules and $goto\ n$ actions might introduce loops, in principle. Our completeness assumption guarantees that the set of rules can always be modelled as a directed acyclic graph (DAG). In other words, the same packet cannot be checked against the same rule more than once (i.e., the evaluation flow does not include loops that would prevent the evaluation process to end, thus violating the completeness requirement).

With a little abuse of terminology in the following we will call fw both the firewall device and its rule set, unless ambiguities might arise. Moreover, we will adopt the dotted notation to identify fields in (conceptually) nested structures, e.g., $fw.r_i.action$ stands for the action associated to the i-th rule of firewall fw whereas $fw.r_i.C$ represents the rule condition, and $fw.r_i.C[j]$ is the j-th field of condition C in the i-th rule of fw.

With the assumptions described above, the result of the filtering process applied to packet P depends on the first matched rule in fw having either $deny$ or $allow$ as the corresponding action, since $goto\ n$ rules, when matched, only postpone the forward/discard decision. The firewall behavior can then be modeled as a function f

$$f : fw,\ \mathcal{P} \cup \{-\} \ \rightarrow\ \mathcal{P} \cup \{-\} \tag{1}$$

which maps each packet into either the packet itself (allow action) or the null packet $-$ (deny action). In this way, $f(fw, -) = -$, as shown in Fig. 2.

With this arrangement, when multiple firewalls are cascaded, the final decision about the packet can simply be obtained by the ordered composition of filtering functions associated to all the firewalls in the sequence. For instance, to know whether a packet P can successfully cross a sequence of firewalls $fw_1, fw_2, \ldots, fw_n$, we compute the cumulative action $f(fw_n, f(fw_{n-1}, \ldots, f(fw_2, f(fw_1, P))))$.

III. The Methodology Step by Step

A. Problem Statement

Given an RST such as the one in Fig. 1a with all nodes properly configured, let us suppose that, for some reason, firewall fw_1 becomes overloaded with packets flowing from the end device d_0, so causing an increased packet processing latency and/or even possible packet losses.

By assuming that the time needed to check a rule is constant and independent of the rule itself, the average firewall processing time for any packet is proportional to the average number of rules checked till a match is found. Therefore, in order to reduce the latency and load introduced by filtering, a suitable set of rules can be moved from fw_1 to some firewall(s) $fw_{2,i}, i \in [1, N]$ to filter packets forwarded to subnetwork(s) $d_{2,i}, i \in [1, N]$.

Let us assume that only $fw_{2,1}, \ldots, fw_{2,K},\ K \in [1, N]$ are involved in such a change of configuration, whereas $fw_{2,K+1}, \ldots, fw_{2,N}$ are left unchanged and define

$$D_1^K = \bigcup_{i=1}^{K} d_{2,i}\,, \qquad D_{K+1}^N = \bigcup_{i=K+1}^{N} d_{2,i}\,. \tag{2}$$

For the sake of simplicity we assume that the whole address and port ranges assigned to D_1^K and D_{K+1}^N can be expressed in a simple way (i.e., by means of regular expressions), that is without the need of splitting them through multiple rules. In other words, the whole ranges in D_1^K and D_{K+1}^N can be specified with simple field formulas. Though this hypothesis could be easily removed without affecting the validity of the proposed solution, it allows a lighter mathematical notation in the following.

Let $\widehat{fw}_1, \widehat{fw}_{2,1}, \ldots, \widehat{fw}_{2,K}$ be the firewall configurations transformed with the purpose of reducing the fw_1 workload at the expense of $fw_{2,1}, \ldots, fw_{2,K}$. The original filtering semantics of the multiple firewall infrastructure must obviously be preserved. To simplify the notation, we rename $fw_{2,i}$ as $\widehat{fw}_{2,i}$ $\forall i \in [K+1, N]$, that is the transformation $fw_{2,j} \rightarrow \widehat{fw}_{2,j}$ $\forall j \in [1, N]$ applies indistinctly to all firewalls, but when $j \in [K+1, N]$ no change is introduced in the firewall configurations. More formally:

Theorem 1: $\forall$ *packet* P *originating from* d_0, $\forall i \in [1, N]$,

$$f(fw_{2,i}, f(fw_1, P)) = f(\widehat{fw}_{2,i}, f(\widehat{fw}_1, P)).$$

This means that each packet coming from d_0 and forwarded to any subnetwork $d_{2,1}, \ldots, d_{2,N}$, must not be affected by the transformation applied, and either it reaches the intended destination or it is discarded in both cases.

Theorem 2: $\forall$ *packet* P *sent from* $d_{2,i}$ *to* $d_{2,j}$, $\forall i, j \in [1, N]$, $i \neq j$,

$$f(fw_{2,j}, P) = f(\widehat{fw}_{2,j}, P).$$

This means that the transformation does not affect packets exchanged among subnetworks $d_{2,1}, \ldots, d_{2,N}$.

Proofs for Theorems 1 and 2 can be found in the following. It is worth observing that the two theorems refer to different streamed topologies, as shown in Fig. 1a (Theorem 1) and in Fig. 1b (Theorem 2) by selecting, for instance, $i = 2$.

B. The REDIAL Algorithm

In the following we postulate that all nodes in the system are properly configured, that is, we assume that:

1) fw_1 does not forward packets from d_0 to d_1 if their source IP addresses belong to d_1, as in the case of trivially malformed or malicious messages;
2) no firewall $fw_{2,j}$ forwards packets from $d_{2,j}$ to d_1 if their source IP addresses belong to d_1 for the same reason as above.

Of course d_1 plays different roles in these two situations, with respect to different flows and RSTs: the first one is rooted in d_0 and $d_1 = \bigcup_i d_{2,i}$ $\forall i \in [1, N]$ (Fig. 1a), whereas in the latter $d_1 = d_0 \cup \bigcup_i d_{2,i}$ $\forall i \in [1, N]$, $i \neq j$, as Fig. 1b shows when $j = 2$ (i.e., the RST is rooted in $d_{2,2}$ and d_0 is a leaf).

The REDIAL algorithm in Fig. 3 transforms the relevant firewall configurations so as to achieve the goal introduced in subsection III-A. According to our model and notation, only firewalls $\widehat{fw}_{2,i}$ $\forall i \in [1, K]$ have to filter packets previously managed by fw_1, and respectively routed to the guarded subnetworks $d_{2,i}$ (domain D_1^K), whereas $fw_{2,i}$ and $d_{2,i}$ $\forall i \in [K+1, N]$ (domain D_{K+1}^N) are not affected by the transformation. As discussed below, REDIAL builds $\widehat{fw}_1$ by simply copying $\overline{fw}_1$ at the end, that is $\widehat{fw}_1$ and $\overline{fw}_1$ contain the same sequence of rules after the algorithm is run.

The outermost loop of REDIAL (lines 1-20 in Fig. 3) scans the rules in fw_1: the IP destination address field d_ip in each rule condition ($C[d\_ip]$) is checked against the IP address range of each subnetwork $d_{2,j}$ protected by firewall $fw_{2,j}$ with $j \in [1, K]$ (lines 2-13). If a match is found (line 3) the rule is added to $\overline{fw}_{2,j}$ in position k_j (line 4). Then the translation table TT is updated by keeping track that the i-th rule in the original fw_1 sequence is now the k_j-th rule in the new configuration of $\overline{fw}_{2,j}$ (line 5). k_j is consequently incremented (line 6).

Lines 7-11 are executed only once: the first time a rule which can match packets to D_1^K is found, a new rule crafted so as to enable all traffic to D_1^K is added to the new configuration of $\overline{fw}_1$. In this way $\widehat{fw}_1$ will forward all packets addressing the relevant subnetwork, without any further check against the following rules in its sequence. This is a key-point to reduce the fw_1 filtering load.

After considering the ability of rule $fw_1.r_i$ to match packets to D_1^K through lines 2-13, lines 14-18 check whether packets addressing D_{K+1}^N can match the rule condition ($fw_1.r_i.C[d\_ip] \cap D_{K+1}^N \neq \emptyset$). If so, the rule is added to $\overline{fw}_1$, i.e., $\widehat{fw}_1$ shall keep on processing the traffic to D_{K+1}^N, whose firewalls are not involved in the transformation. TT is also updated to keep track of the correspondence between the positions of the rule in fw_1 and $\overline{fw}_1$.

In copying a rule $fw_1.r_i$ from fw_1 to any $\overline{fw}$, the destination IP address field $C[d\_ip]$ does not need to be modified, as the rule placed in $\overline{fw}$ will actually match only a proper subset of all packets originally matched when it was located in fw_1.

The two nested loops at lines 21-28 in Fig. 3 update the destination labels of *goto n* rules. For each fw_1 goto rule copied to $\overline{fw}$, the new value for index n is found (line 24) and the rule updated accordingly (line 25).

At the end, $\overline{fw}_1$ is trivially copied to $\widehat{fw}_1$ (line 29), whereas the construction of $\widehat{fw}_{2,i}$ $\forall i \in [1, K]$ (lines 30 - 47) is a bit more tricky because each $\widehat{fw}_{2,i}$ must:

- Act the same way as fw_1 on packets received from $\widehat{fw}_1$ and addressing $d_{2,i}$. This means
  - to discard those packets to $d_{2,i}$ that would be dropped by fw_1;
  - to check against the same $fw_{2,i}$ sequence those packets to $d_{2,i}$ that would be forwarded by fw_1.
- Check against the same $fw_{2,i}$ sequence all packets received from any $d_{2,j}$ $j \neq i$.

In other words, $\widehat{fw}_{2,i}$ has to perform different actions for each case above, and this is taken into account by lines 31-46 of REDIAL in Fig. 3. A new *goto* $|\overline{fw}_{2,i}|+2$ rule is placed at the top of the $\widehat{fw}_{2,i}$ sequence, so that packets not originating from d_0 (this is expressed as $any \backslash d_0$ at line 31 of the algorithm) can skip the following $|\overline{fw}_{2,i}|$ rules. It is worth noting that if $any \backslash d_0$ cannot be specified through a single regular expression, at most two rules are needed at the top of $\widehat{fw}_{2,i}$. In this case, the first rule simply skips the second one when the source address of the incoming packet is d_0, whereas the second jumps unconditionally to the first rule that is obtained from the original $fw_{2,i}$ sequence. Then the $\overline{fw}_{2,i}$ rules are copied to $\widehat{fw}_{2,i}$ starting at position 2. In doing this *deny* actions are left unchanged, whereas *allow* actions are transformed into *goto* $|\overline{fw}_{2,i}|+2$, so that packets originally forwarded by fw_1 are still checked against the $fw_{2,i}$ sequence. Finally, the $fw_{2,i}$ rules are appended to the resulting firewall starting at position $|\overline{fw}_{2,i}|+2$. During the process, goto indexes for all rules copied from both $\overline{fw}_{2,i}$ and $fw_{2,i}$ are suitably updated.

As a last step, $fw_{2,i} \rightarrow \widehat{fw}_{2,i}$ $\forall i \in [K+1, N]$.

The updated firewall $\widehat{fw}_1$ might not have fewer rules than fw_1, in fact $1 \leq |\widehat{fw}_1| \leq |fw_1| + 1$. When $|\widehat{fw}_1| = 1$

```
Input:  $fw_1$   $fw_{2,i}$ $\forall i \in [1,N]$   $d_1$   $d_{2,i}$ $\forall i \in [1,N]$
        $D_1^K$ and $D_{K+1}^N$ computed by(2) from all $d_{2,i}$

Output: updated firewalls $\widehat{fw}_1$, $\widehat{fw}_{2,j}$ $\forall j \in [1,N]$

$\widehat{fw}$s and $\overline{fw}$s are all initially empty
$flag$ is initially set to FALSE
 $i$ ranges over $fw_1$   $j \in [1,K]$ ranges over $d_{2,1}, \dots d_{2,K}$              all these indexes are
 $h$ ranges over $\overline{fw}_1$   $k_1, \dots, k_K$ ranges respectively over $\overline{fw}_{2,1}, \dots, \overline{fw}_{2,K}$   initially set to 1
$TT[|fw_1|, K+1]$ is a Translation Table needed to adjust the destinations of the goto actions
by taking into account that, in general, only a subset of the rules of $fw_1$ goes to some $\overline{fw}$.
$TT[i, \overline{fw}]$ stores the position in $\overline{fw}$ of the $i$-th rule of $fw_1$. $TT$ is initially set to 0.

1: while( $i \le |fw_1|$ ) {
2:   while( $j \le K$ ) {
3:     if( $fw_1.r_i.C[d\_ip] \cap d_{2,j} \ne \emptyset$ ) {
4:       add $fw_1.r_i$ to $\overline{fw}_{2,j}$ as $r_{k_j}$;
5:       $TT\left[i, \overline{fw}_{2,j}\right] \leftarrow k_j$;
6:       $k_j++$;
7:       if( $flag$ = FALSE ) {
8:         $flag \leftarrow$ TRUE;
9:         add $(\mathtt{any}, \mathtt{D_1^K}, \mathtt{any}, \mathtt{any}, \mathtt{any}, \mathtt{accept})$ to $\overline{fw}_1$ as $r_h$;
10:        $h++$;
11:      }
12:    }
13:  }
14:  if( $fw_1.r_i.C[d\_ip] \cap D_{K+1}^N \ne \emptyset$ ) {
15:    add $fw_1.r_i$ to $\overline{fw}_1$ as $r_h$;
16:    $TT\left[i, \overline{fw}_1\right] \leftarrow h$;
17:    $h++$;
18:  }
19:  $i++$;
20:}
21:$\forall$ $column$ $\overline{fw}$ of $TT$ {
22:  $\forall$ $row$ $i$ of $TT$ {
23:    if( $TT[i, \overline{fw}] \ne 0 \ \wedge \ \overline{fw}.r_{TT[i,\overline{fw}]}.action = \mathtt{goto\ n}$ ) {
24:      for( $j \leftarrow \mathtt{n}$; $TT[j, \overline{fw}] = 0$; $j++$);
25:      $\overline{fw}.r_{TT[i,\overline{fw}]}.action \leftarrow \mathtt{goto\ j}$;
26:    }
27:  }
28:}
29:$\widehat{fw}_1 \leftarrow \overline{fw}_1$;
30:for( $i \leftarrow 1$; $i \le K$; $i++$) {
31:  $\widehat{fw}_{2,i}.r_1 \leftarrow (\mathtt{any} \setminus \mathtt{d_0}, \mathtt{any}, \mathtt{any}, \mathtt{any}, \mathtt{any}, \mathtt{goto}|\overline{\mathtt{fw}}_{2,i}| + 2)$;
32:  for( $j \leftarrow 2$; $j \le |\overline{fw}_{2,i}| + 1$; $j++$) {
33:    if( $\overline{fw}_{2,i}.r_{j-1}.action = \mathtt{goto\ n}$) {
34:      $\overline{fw}_{2,i}.r_{j-1}.action \leftarrow \mathtt{goto\ n+1}$;
35:    }
36:    else if( $\overline{fw}_{2,i}.r_{j-1}.action = \mathtt{allow}$) {
37:      $\overline{fw}_{2,i}.r_{j-1}.action \leftarrow \mathtt{goto}\ |\overline{\mathtt{fw}}_{2,i}| + 2$;
38:    }
39:    $\widehat{fw}_{2,i}.r_j \leftarrow \overline{fw}_{2,i}.r_{j-1}$;
40:  }
41:  for( ; $j \le |\overline{fw}_{2,i}| + |fw_{2,i}| + 1$; $j++$) {
42:    if( $fw_{2,i}.r_{j-|\overline{fw}_{2,i}|-1}.action = \mathtt{goto\ n}$) {
43:      $fw_{2,i}.r_{j-|\overline{fw}_{2,i}|-1}.action \leftarrow \mathtt{goto\ n} + |\overline{\mathtt{fw}}_{2,i}| + 1$;
44:    }
45:    $\widehat{fw}_{2,i}.r_j \leftarrow fw_{2,i}.r_{j-|\overline{fw}_{2,i}|-1}$;
46:  }
47:}
48:for( ; $i \le N$; $i++$) {
49:  $\widehat{fw}_{2,i} \leftarrow fw_{2,i}$;
50:}
```

Fig. 3. The REDIAL Algorithm.

all fw_1 rules can match packets to D_1^K, (i.e., the action at line 9 is always performed), and no packet to D_{K+1}^N can be matched (the result of test at line 14 is always false). Instead, when $|\widehat{fw}_1| = |fw_1| + 1$ all rules in fw_1 can match packets to both D_1^K and D_{K+1}^N. This situation does not imply worse performance. Indeed, our proposal, besides relying on the reduction of the number of rules checked in fw_1, also forces all packets to D_1^K to be matched by exactly one rule in $\widehat{fw}_1$ (line 9). The index h of the new rule in $\widehat{fw}_1$ cannot be higher than the index of first rule possibly matching packets to D_1^K in fw_1. In other words, all packets to D_1^K are matched by the h-th rule in $\widehat{fw}_1$, whereas the same packets are matched by rules placed somewhere between h and $|fw_1|$ in fw_1.

It is worth observing that when D_1^K consists of $k \leq K$ separate sub-domains, at most k rules have to be inserted in $\widehat{fw}_1$ (line 9). The performance impact of this change is negligible as K is very small in most practical situations. Moreover the k rules can be placed starting at any position between 1 and h in $\widehat{fw}_1$, as they are totally independent from the top h-1 rules in fw_1. The insertion of rules starting at position h is conservative with respect to the maximum performance improvement that can be obtained with our solution.

Let us now introduce some properties needed to prove Theorems 1 and 2. In order to do so, we refer to the RST in Fig. 1a, the firewall function (1) depicted in Fig. 2, and the REDIAL algorithm.

Property 1: ∀ packet P originating from d_0 and addressing D_1^K,

$$f(\widehat{fw}_1, P) = P\,.$$

Proof: By inspecting the algorithm in Fig. 3, there is no rule in $\overline{fw}_1$ which can match P and precedes the rule added to $\overline{fw}_1$ at line 9. As the action associated to this new rule is `allow`, the firewall function in Fig. 2 returns P. □

Property 2: Rule $fw_1.r$ can be matched by some packets addressing subnetwork $d_{2,i}$, $\forall i \in [1, K]$ ⇔ rule $fw_1.r$ is added to $\overline{fw}_{2,i}$.

Proof: By inspecting the algorithm in Fig. 3:

⇒ If the test at line 3 is true, i.e. the rule condition matches packets addressing $d_{2,i}$, then the rule is added to $\overline{fw}_{2,i}$.

⇐ A rule of fw_1 is added to $\overline{fw}_{2,i}$ (line 4) only if it can match packets addressing $d_{2,i}$ (line 3). There are not instances of this action which are not guarded by a test on the destination address. □

Property 3: ∀ packet P originating from d_0 and addressing $d_{2,i}$, $\forall i \in [1, K]$,

$$f(fw_1, P) = f(\overline{fw}_{2,i}, P)\,.$$

Proof: By Property 2, $\overline{fw}_{2,i}$ contains only the rules in fw_1 which are able to match packets to $d_{2,i}$. Moreover, by inspecting the algorithm in Fig. 3, these rules are added to $\overline{fw}_{2,i}$ in the same relative order as they are met in fw_1. This means that, from the point of view of a packet P addressed to $d_{2,i}$, the difference between fw_1 and $\overline{fw}_{2,i}$ consists only of rules that can never be matched, causing function f in Fig. 2 to produce the same result when P is checked against the two sequences in fw_1 and $\overline{fw}_{2,i}$. The completeness of fw_1 guarantees that such a result always exists, i.e., $\overline{fw}_{2,i}$ is complete for packets addressing $d_{2,i}$. □

Property 4: Rule $fw_1.r$ can be matched by some packets addressing subnetwork(s) in D_{K+1}^N ⇔ rule $fw_1.r$ is added to $\overline{fw}_1$.

Proof: The same proof as for Property 2, with lines 3 and 4 of the algorithm replaced by lines 14 and 15 respectively. □

Property 5: ∀ packet P originating from d_0 and addressed to D_{K+1}^N,

$$f(fw_1, P) = f(\overline{fw}_1, P)\,.$$

Proof: The same proof as for Property 3, with Property 2 replaced by Property 4. □

Properties 1, 4, and 5 also hold for $\widehat{fw}_1$, whereas Properties 2 and 3 are true for the subset of rules in $\widehat{fw}_{2,i}$ that were added to $\overline{fw}_{2,i}$ during its construction.

Properties 2 and 4 imply that some rules of fw_1 may not be copied to some $\overline{fw}$. The first condition of the test at line 23 in Fig. 3 means that if $TT[i, \overline{fw}] = 0$, the i-th rule of fw_1 has not been copied to $\overline{fw}$, so the second condition of the test becomes meaningless as there is no corresponding rules. For the same reason, also the original destination of a goto action may also not be present in $\overline{fw}$. This is why the rule following the missing jump destination is searched at line 24. The completeness assumption guarantees that such a rule exists and belongs to the set of rules reachable through the goto action. In fact, a packet matching the goto condition, without matching any other rule in the destination block, would violate the completeness hypothesis.

Property 6: ∀ packet P originating from d_0 and addressing $d_{2,i}$, $\forall i \in [1, K]$,

$$f(fw_{2,i}, f(fw_1, P)) = f(\widehat{fw}_{2,i}, P)\,.$$

Proof: Property 3 both allows to change the theorem claim to $f(fw_{2,i}, f(\overline{fw}_{2,i}, P)) = f(\widehat{fw}_{2,i}, P)$ while guaranteeing that $\overline{fw}_{2,i}$ is complete with respect to P, i.e., a rule surely exists in $\overline{fw}_{2,i}$ whose action *deny* or *allow* matches P.

Since P comes from d_0, the first rule of $\widehat{fw}_{2,i}$ cannot be matched and P is checked against the subsequent rules. Rules 2 to $|\overline{fw}_{2,i}| + 1$ in $\widehat{fw}_{2,i}$ are copied from $\overline{fw}_{2,i}$, by only changing any *allow* action to `goto` $|\overline{fw}_{2,i}| + 2$. This means that P in $\overline{fw}_{2,i}$ can only be one of the following two:

1) *deny*, then the same happens in $\widehat{fw}_{2,i}$, thus proving the property because $f(\overline{fw}_{2,i}, P) = -$ leads to $f(fw_{2,i}, f(\overline{fw}_{2,i}, P)) = -$, exactly as $f(\widehat{fw}_{2,i}, P)$ does;
2) *allow*, then $f(\overline{fw}_{2,i}, P) = P$ and the claim becomes $f(fw_{2,i}, P) = f(\widehat{fw}_{2,i}, P)$. Moreover the action of the corresponding rule in $\widehat{fw}_{2,i}$ is `goto` $|\overline{fw}_{2,i}|$+2 causing function f to keep on checking P in $\widehat{fw}_{2,i}$ against the rules coming from $fw_{2,i}$. This makes the new claim $f(fw_{2,i}, P) = f(\widehat{fw}_{2,i}, P)$ true.

The completeness of $\overline{fw}_{2,i}$ guarantees that no other alternative is possible. □

Theorems 1 and 2 in Section III-A can now be proved.

Proof of Theorem 1: First of all note that only packets addressing $d_{2,i}$ have to be taken into account for each i, as packets not addressed to $d_{2,i}$ make the theorem meaningless. Then, let us distinguish the following two cases, depending on the packet destination subnetwork:

1) $i \in [1, K]$: Property 1 states that $f(\widehat{fw}_1, P) = P$; it follows that $f(fw_{2,i}, f(fw_1, P)) = f(\widehat{fw}_{2,i}, P)$, i.e., the claim of Property 6.
2) $i \in [K + 1, N]$ means that $d_{2,i} \in D_{K+1}^N$ and $\widehat{fw}_{2,i} = fw_{2,i}$, hence the theorem claim becomes

$f(fw_{2,i}, f(fw_1, P)) = f(fw_{2,i}, f(\widehat{fw}_1, P))$, where $f(fw_1, P) = f(\widehat{fw}_1, P)$ holds by Property 5, because $\widehat{fw}_1$ *is* $\overline{fw}_1$.

Proof of Theorem 2: Note that only packets addressing $d_{2,j}$ have to be taken into account for each j. Packets not addressed to $d_{2,j}$ make the theorem meaningless. Let us distinguish the following two cases, depending on the packet destination subnetwork:

1) $j \in [1, K]$: because of the construction of $\widehat{fw}_{2,j}$ any packet from $d_{2,i}$ matches the first rule of $\widehat{fw}_{2,j}$ and skips the following block of rules copied from fw_1 through $\overline{fw}_{2,i}$. The packet is then checked against the rules starting at position $|\overline{fw}_{2,j}| + 2$, that were originally in $fw_{2,j}$. In this way the packet is checked against the same set of rules both before and after the transformation.
2) $j \in [K+1, N]$: in this case $d_{2,j} \in D_{K+1}^N$, i.e., $\widehat{fw}_{2,j}$ is $fw_{2,j}$ and the theorem claim becomes $f(fw_{2,j}, P) = f(fw_{2,j}, P)$.

IV. Transformation Performance

A. Performance Bounds

Performance of the proposed technique can be estimated by means of coarse-grained data, that is easily obtained by run-time monitoring of fw_1, and some parameters computed with the REDIAL algorithm in Fig. 3. Indeed, predicting the effectiveness of the transformation is useful in a number of circumstances, including the design of the network protection infrastructure or the evaluation of actions to be taken in response to threats like denial-of-service attacks. In this section we provide bounds for the improvement in the packet processing rate of $\widehat{fw}_1$ with respect to fw_1, by taking into account some characteristics of firewall configurations, traffic profile and system architecture. In particular, our performance model relies on:

- The number of rules in fw_1 and $\widehat{fw}_1$, i.e. $|fw_1|$ and $|\widehat{fw}_1|$ respectively.
- The fraction ρ $(0 \le \rho \le 1)$ of packets sent from d_0 to D_1^K, and the corresponding fraction $(1-\rho)$ of packets from d_0 to D_{K+1}^N.
- The average number σ_1^K of rules evaluated by fw_1 for each packet addressed to D_1^K.
- The average number $\widehat{\sigma}_1^K$ of rules evaluated by $\widehat{fw}_1$ (that is after the transformation) for each packet addressed to D_1^K.
- The average number σ_{K+1}^N of rules evaluated by fw_1 for each packet addressed to D_{K+1}^N.
- The average number $\widehat{\sigma}_{K+1}^N$ of rules evaluated by $\widehat{fw}_1$ (after the transformation) for each packet addressed to D_{K+1}^N.
- The position h of the rule added to $\widehat{fw}_1$ at line 9 of the algorithm in Fig. 3, which forwards all packets addressed to D_1^K.

Given D_1^K and D_{K+1}^N in (2), ρ, σ_1^K and σ_{K+1}^N can be obtained by direct observation of the traffic profile from d_0 and the fw_1 behavior during a suitable timeframe. In addition, both h and $|\widehat{fw}_1|$ can be computed by running the REDIAL algorithm in Fig. 3, whereas $|fw_1|$ is known a priori. Then the average number of rules evaluated by fw_1 for each packet received from d_0 is

$$\sigma_1^N = \sigma_1^K \cdot \rho + \sigma_{K+1}^N \cdot (1-\rho)\,, \tag{3}$$

while the average number of rules evaluated by $\widehat{fw}_1$ for each packet received from d_0 due to the transformation is

$$\widehat{\sigma}_1^N = h \cdot \rho + \widehat{\sigma}_{K+1}^N \cdot (1-\rho)\,, \tag{4}$$

in fact, $\widehat{\sigma}_1^K = h$, as each packet addressed to D_1^K requires the evaluation of exactly h rules by $\widehat{fw}_1$.

Given σ_1^N and $\widehat{\sigma}_1^N$, we define

$$\Gamma = \frac{\sigma_1^N}{\widehat{\sigma}_1^N} \tag{5}$$

the *performance gain* of our transformation. Since we assumed that the time needed to evaluate a rule is independent of the rule itself, Γ represents the ratio between the packet processing rates of $\widehat{fw}_1$ and fw_1. For instance, if $\sigma_1^N = 10$ and $\widehat{\sigma}_1^N = 2$, $\Gamma = 5$ means that $\widehat{fw}_1$ is five times faster than fw_1 on average.

The computation of (5) relies on $\widehat{\sigma}_{K+1}^N$ in (4), which cannot be measured unless either $\widehat{fw}_1$ is actually deployed or the same traffic statistics collected for fw_1 are used to simulate the $\widehat{fw}_1$ behavior after running REDIAL.

Nevertheless, the value of Γ can be estimated by computing the best, worst and average case for $\widehat{\sigma}_{K+1}^N$.

The difference between σ_{K+1}^N and $\widehat{\sigma}_{K+1}^N$ depends on both the number of rules matching packets to D_1^K (they are not copied to $\widehat{fw}_1$) and how the rules themselves are distributed in fw_1. The following situations have to be considered for rules matching D_1^K :

- Rules are all placed at the top of the fw_1 sequence:

$$\widehat{\sigma}_{K+1}^N = \sigma_{K+1}^N - \left(|fw_1| - |\widehat{fw}_1|\right)\,. \tag{6}$$

- Rules are evenly distributed in the fw_1 sequence:

$$\widehat{\sigma}_{K+1}^N = \sigma_{K+1}^N \cdot \frac{|\widehat{fw}_1|}{|fw_1|}\,. \tag{7}$$

- Rules are all placed at the bottom of the fw_1 sequence, so that they do not affect $\widehat{\sigma}_{K+1}^N$ with respect to σ_{K+1}^N except for the rule added at line 9 of algorithm in Fig. 3:

$$\widehat{\sigma}_{K+1}^N = \sigma_{K+1}^N + 1\,. \tag{8}$$

Since $\sigma_{K+1}^N \le |fw_1|$, it follows that:

$$\sigma_{K+1}^N - \left(|fw_1| - |\widehat{fw}_1|\right) \;\le\; \sigma_{K+1}^N \cdot \frac{|\widehat{fw}_1|}{|fw_1|} \;\le\; \sigma_{K+1}^N + 1 \tag{9}$$

and (6) and (8) are respectively lower and upper bounds for $\widehat{\sigma}_{K+1}^N$, whereas (7) is the average value when rules are evenly distributed in the firewall sequence and the input packet flow has a uniform distribution.

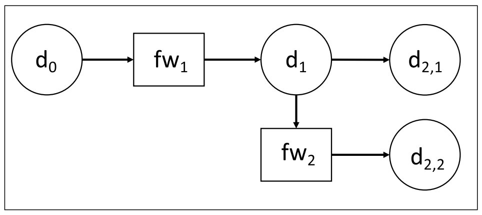

Fig. 4. Simple network RST adopted in the simulation experiments.

By substituting (6), (7) and (8) in (4), equation (5) for the three situations considered becomes:

$$\Gamma_M = \frac{\sigma_1^K \cdot \rho \ + \ \sigma_{K+1}^N \cdot (1-\rho)}{h \cdot \rho \ + \ \left(\sigma_{K+1}^N - |fw_1| + |\widehat{fw}_1|\right) \cdot (1-\rho)}, \quad (10)$$

$$\Gamma_{av} = \frac{\sigma_1^K \cdot \rho \ + \ \sigma_{K+1}^N \cdot (1-\rho)}{h \cdot \rho \ + \ \left(\sigma_{K+1}^N \cdot \frac{|\widehat{fw}_1|}{|fw_1|}\right) \cdot (1-\rho)}, \quad (11)$$

$$\Gamma_m = \frac{\sigma_1^K \cdot \rho \ + \ \sigma_{K+1}^N \cdot (1-\rho)}{h \cdot \rho \ + \ \left(\sigma_{K+1}^N + 1\right) \cdot (1-\rho)}, \quad (12)$$

giving respectively the maximum, average and minimum possible value for the transformation performance gain Γ. In normal conditions, and with a reasonable amount of traffic ρ addressing D_1^K, σ_1^K is significantly larger than h. This means that the minimum performance gain Γ_m is greater than one and the proposed transformation is always advantageous, at least in theory.

B. Performance Simulation

Correctness and performance of the proposed solution were also verified by means of simulation. A suitable program was then designed and developed in C language, which is able to take into account different configurations for two cascaded firewalls fw_1 and fw_2 and the simple network RST shown in Fig. 4. In the experiments, fw_1 analyzes all packets received from d_0 and addressing either $d_{2,1}$ or $d_{2,2}$, while fw_2 protects the subnetwork $d_{2,2}$ only. In this condition $d_{2,1}$ plays the role of D_{K+1}^N whereas $d_{2,2}$ represents D_1^K.

The simulator enables the selection of several parameters, including the number of rules in fw_1 and fw_2, the percentage of *allow*, *deny* and *goto n* actions (α_a, α_d and α_g respectively), the percentages (β_1, β_2 and β_{12}) of fw_1 rules concerning $d_{2,1}$, $d_{2,2}$ or both and so on.

Packets originating from d_0 are randomly generated with a uniform distribution for any of the five fields of interest, that is source and destination IP addresses, source and destination ports and protocol type. The percentage of packets addressing $d_{2,1}$ and $d_{2,2}$ can also be selected as requested in the experiments.

The correctness of the model and algorithm in Fig. 3 was preliminary and exhaustively checked for a network with reduced address ranges, adopting 10 bits for each IP address and 5 bits for each port number and protocol type. In particular, all the 2^{35} possible packets were fed to 500 different configurations randomly selected for the two firewalls, both before and after their transformation. The comparison of the resulting actions performed on each packet in the two conditions confirmed that the input traffic is treated absolutely the same way by the (fw_1,fw_2) and ($\widehat{fw}_1$,$\widehat{fw}_2$) pairs as expected. Subsequently, experiments were conducted to check the performance improvement that can be obtained with our approach. To this purpose several configurations were tested by keeping the number of rules in fw_1 and fw_2 constant, while changing their conditions $r_i.C$ and actions $r_i.action$. The input load originating from d_0 in each experiment consisted of a set of one million randomly-generated packets for each point of the plots in Fig. 5 and Fig. 6, where the percentage of packets addressing $d_{2,2}$ was varied from 0% to 100% with 1% increments.

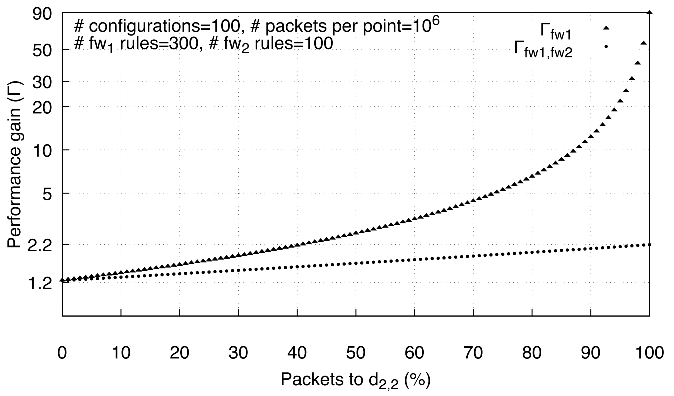

Fig. 5. Performance gain obtained by simulation of 100 firewall configurations.

Fig. 5 shows the performance behavior (logarithmic y-axis scale) obtained by averaging the simulation results for 100 different configurations with 300 rules in fw_1 ($\alpha_d = 35\%$, $\alpha_a = 60\%$, $\alpha_g = 5\%$, $\beta_1 = 60\%$, $\beta_2 = 20\%$, $\beta_{12} = 20\%$) and 100 rules in fw_2. The plot Γ_{fw1} in Fig. 5 confirms that the relative improvement in performance for fw_1 ranges from about 1.2 when no packet is sent to $d_{2,2}$ and each message addresses $d_{2,1}$, to 90 when the whole offered load targets $d_{2,2}$ (100%). It is worth observing that $\Gamma_{fw_1} \geq 1$ even if no traffic is forwarded to $d_{2,2}$, since packets addressing $d_{2,1}$ are checked by $\widehat{fw}_1$ against a smaller number of rules on average. However, as expected, the most benefit is obtained when destinations of all packets from d_0 are located in $d_{2,2}$.

The overall performance gain Γ_{fw_1,fw_2} for the ($\widehat{fw}_1$,$\widehat{fw}_2$) pair is obviously lower and ranges from 1.2 to 2.2, as the decrease in the $\widehat{fw}_1$ filtering activity is partially compensated by the increment in rule processing by $\widehat{fw}_2$.

Of course, the actual value that can be obtained for Γ heavily depends on the characteristics of the firewall configurations and the input traffic profile, thus it has to be evaluated on a case by case basis. For instance, plots in Fig. 6 show the behavior of Γ_{fw1} and Γ_{fw_1,fw_2} when 500 configurations are considered and 50 rules are set for both fw_1 and fw_2 ($\alpha_d = 45\%$, $\alpha_a = 45\%$, $\alpha_g = 10\%$, $\beta_1 = 40\%$, $\beta_2 = 40\%$, $\beta_{12} = 20\%$). In this condition the improvement for $\widehat{fw}_1$ falls between 1.6 and 28, whereas a small decrease (from 1.6 to 1.4) is observed for the ($\widehat{fw}_1$, $\widehat{fw}_2$) pair, when the percentage of packets addressing $d_{2,2}$ gets close to 100, but Γ_{fw_1,fw_2} is always kept greater than one.

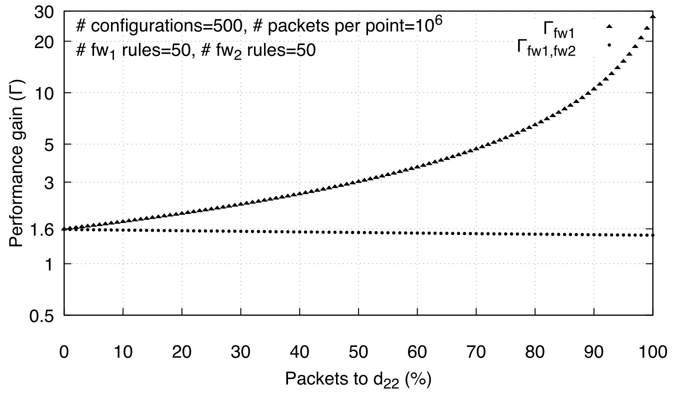

Fig. 6. Performance gain obtained by simulation of 500 firewall configurations with 50 rules each.

C. Experimental Results

Performance in Section IV-B has been derived in steady state conditions, by ideally assuming that switching of the firewall configurations can be done in negligible time. The experimental study of transient conditions triggered by the transformation deployment, instead, can shed some light on situations that can hardly be observed through simulation. In particular, we are interested in measuring the time needed to perform the transformation from (fw_1, fw_2) to $(\widehat{fw_1}, \widehat{fw_2})$ in a realistic scenario and getting some estimations of its impact on the filtered traffic in terms of packet filtering delay. To this purpose a test-bed was setup in our Industrial Informatics Laboratory (I^2Lab), also leveraging past experience with firewall modeling, characterization and performance evaluation [3], [28], [35].

The structure of the simple network depicted in Fig. 4 was then adopted for the test-bed and timing data collected by means of suitable traffic access points (TAPs) and cards supporting hardware timestamps.

Fig. 7 shows the test-bed set-up composed by:

- The traffic generator G (d_0) is an Apple iMac equipped with an Intel® Core™ 2 Extreme X7900 CPU @ 2.80GHz, 4 GB of RAM and running the 20.04 LTS Ubuntu Linux distribution.
- Firewalls FW_1 and FW_2 are personal computers, each one equipped with an Intel® Core™ 2 Duo E6750 CPU @ 2.66GHz, 4GB of RAM and two identical Realtek Semiconductor NICs (ICSs RTL-8139). They run the 18.04 LTS Ubuntu Linux distribution with the iptables firewall configurator.
- Receiver R is a PC with configuration similar to FW_1 and FW_2, where NICs are replaced with a four-ports Intel® Ethernet Server Adapter I350 [36], which is able to hardware timestamp the incoming packets.
- TAP_1 and TAP_2 are Keysight Technologies ® TP-CU3-ST, 10/100/1000 Copper Taps [37].

Packets sent to d_1 are routed to either $d_{2,1}$ or $d_{2,2}$ (which are not relevant to our experiments) by means of SW, an industrial Belden® MACH104-20TX-F managed switch [38].

With these arrangements, TAP_1 duplicates packets from d_0, whereas TAP_2 does the same with packets forwarded by FW_1. Duplicated streams are processed by R, where the Ethernet adapter timestamps each incoming packet. Then the time needed by FW_1 to manage each packet can be easily computed as the difference of two timestamps. By suitably positioning TAP_1 and TAP_2 the performance of either FW_2 or the (FW_1, FW_2) pair can also be measured.

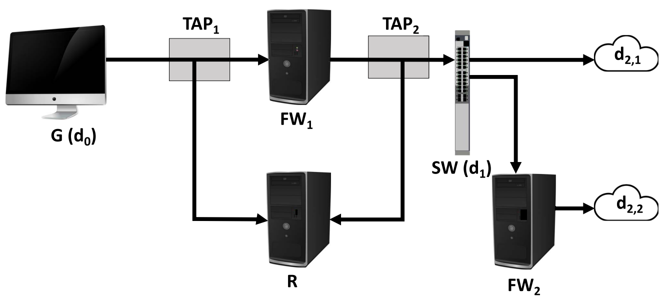

Fig. 7. I^2Lab test-bed for performance evaluation.

In conducting experiments we focused on the following aspects:

- The average packet delay introduced by the filtering activity of FW_1 and FW_2 before and after the transformation.
- How the transformation deployment affects the packet delay, i.e. the overall filtering performance. It is worth noting that the impact of the reconfiguration can be assessed without considering the time needed to run the REDIAL algorithm of Fig. 3, because this task can be performed offline for different traffic profiles, as the firewall rules and the network topology are known and static. Alternatively, REDIAL can be executed on a node different from FW_1 and FW_2, so that it has no impact on their performance.
- The penalty paid in terms of communication bandwidth consumption by packets flowing from d_0 to FW_2 before and after the transformation.

As the performance gain depends on the percentage ρ of traffic addressing $d_{2,2}$, in the experiments we adopted the same traffic patterns of Fig. 5 with ρ equal to 75%, 50% and 25% respectively. Of course, to obtain the highest benefits, the network should be designed so that D_1^K is the destination of most traffic coming from d_0, corresponding to the rightmost part of the diagrams in Fig. 5. In all experiments FW_1 and FW_2 were loaded with 300 and 100 rules respectively, by selecting some of the sets also used for simulations in Fig. 5 and the same values for parameters α_d, α_a, α_g, β_1, β_2, and β_{12}.

Fig. 8 shows the time evolution of the latency experienced by each packet traversing FW_1 (configured with a 300 rule set), when the percentage of traffic addressing $d_{2,2}$ is 75%. The diagram focuses on the transformation impact on the filtering activity and highlights the increment/decrement of the average latency measured with respect to the initial steady state condition. The figure was drawn using a logarithmic scale for the vertical axis, as the transitory peak (l_p) is too high with respect to the steady state levels (l_1 and l_2). In particular, in the interval between 0 and t_s FW_1 runs its initial configuration fw_1, exhibiting an average filtering latency $l_1 \approx 1.86\mu s$ per packet. At time t_s the reconfiguration is triggered by a

TABLE I
PERFORMANCE MEASURES OBTAINED WITH THE EXPERIMENTAL TEST-BED

			FW_1				FW_1, FW_2				
rule set	ρ	h	σ_1^N	$\widehat{\sigma}_1^N$	Γ_{FW_1}	l_1/l_2	σ_1^N	$\widehat{\sigma}_1^N$	Γ_{FW_1,FW_2}	l_1/l_2	Δ b/w
	75%		217.34	37.73	5.76	5.81	229.48	123.03	1.87	1.91	3.93
1	50%	2	207.19	73.40	2.82	2.96	215.33	130.30	1.65	1.75	3.91
	25%		197.08	109.08	1.81	1.91	201.13	137.51	1.46	1.55	3.92
	75%		228.41	36.95	6.18	5.88	242.27	132.58	1.83	1.92	3.16
2	50%	7	212.91	72.87	2.92	2.78	222.16	136.66	1.62	1.54	3.16
	25%		197.45	108.93	1.81	1.85	202.09	140.81	1.44	1.48	3.14
	75%		201.80	40.77	4.95	5.23	224.05	130.58	1.72	1.81	2.47
3	50%	6	198.10	79.57	2.49	2.48	212.94	139.45	1.53	1.47	1.79
	25%		194.57	118.48	1.64	1.76	201.97	148.41	1.36	1.41	1.79

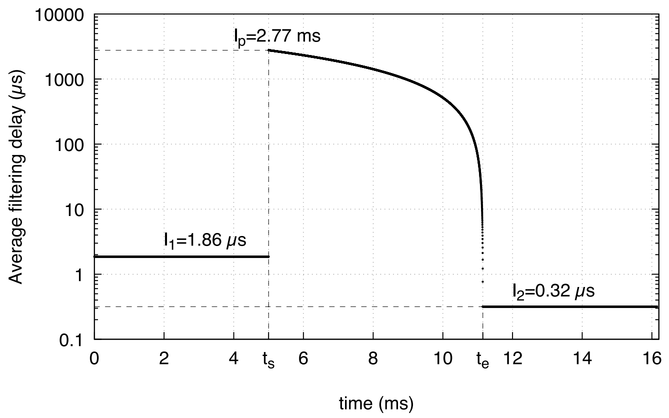

Fig. 8. Firewall filtering latency in steady state and transient conditions.

task running on the same PC, and a transient condition is started lasting till t_e. In our test-bed the transition interval $t_e - t_s$ takes about $6.11ms$ before a new steady state is reached. At t_s the filtering latency suddenly rises by reaching a peak $l_p \approx 2.77ms$, likely because the firewall stops/slows processing packet during the rule tables update. After t_e the new configuration $\widehat{fw}_1$ is fully operational, with a latency average value $l_2 \approx 0.32\mu s$, which is lower than l_1 as expected.

The average number of rules σ_1^N processed by FW_1 before applying the transformation and computed with (3) is approximately 217.34, whereas the corresponding $\widehat{\sigma}_1^N$ value for the $\widehat{fw}_1$ configuration is about 37.73. This means a gain $\Gamma_{FW_1} \approx$ 5.76 (equation 5), in good accordance with the 5.25 value obtained in Fig. 5 and also confirmed by the ratio $l_1/l_2 \approx 5.81$ in Fig. 8.

Experiments were repeated several times without changing the number of rules and the α and β parameters, but with different rule sets as done for simulation. Table I shows the collected measures for three different sets of rules and ρ values. In all conditions the computed value of Γ_{FW_1} is reasonably close to the measured ratio l_1/l_2. For each set Γ_{FW_1} increases as ρ gets larger since $\widehat{\sigma}_1^N$ is reduced. This confirms the trend exhibited by the diagram in Fig. 5. In fact, packets forwarded to $d_{2,2}$ impact more on the average number of checked rules as ρ grows. After the transformation they always match the rule in position h, and this sort of privileged processing contributes to progressively decrease the value of $\widehat{\sigma}_1^N$.

The five rightmost columns in Tab I concern the performance obtained for the (FW_1, FW_2) cascaded pair and report the average number of processed rules per packet before/after the transformation and the resulting computed gain Γ_{FW_1,FW_2} together with the measured l_1/l_2 ratio. It is worth noting that the overall gain slightly decreases with ρ because of the lower advantages obtained for FW_1 with respect to the loss of performance introduced in FW_2.

Finally the last column in the table shows the communication bandwidth penalty (Δ b/w) paid with our solution, caused by the rise of traffic along the path connecting FW_1 to FW_2. The column contains relative values, that is the ratio between the numbers of extra packets reaching FW_2 from d_0 after the transformation and the number of packet forwarded to FW_2 by FW_1 when the set of rules of the latter is fw_1. This metric takes into account only the impact of the rule redistribution and does not consider possibly different phenomena.

V. RELATED WORKS

Several solutions have been proposed in the scientific literature to improve the performance of firewall packet filtering operations. They can roughly be divided into two main groups, that is techniques either operating on a single firewall or relying on network architectures involving multiple firewalls.

A. Intra-Firewall Solutions

In this group three different approaches can be identified, i.e., rule ordering, rule compression and rule analysis. Rule ordering and compression algorithms are able to preserve the firewall integrity while changing the relative position of rules and/or their form/cardinality. Their goal is to improve the device performance, by assuming that the rule set is already compliant with the overall security policy.

Rule analysis, instead, may change the firewall semantics. The main goal, in this case, is to discover weaknesses, errors and misconfigurations, whose fixing might, in principle, make the firewall operation closer to the expected behavior.

1) Rule Ordering: In [14] Hamed *et al.* observed that, in many enterprise networks, a big fraction of the incoming traffic matches only a small subset of the firewall rules, and this subset keeps on being unchanged for long time periods. Then, to improve the average packet filtering time, the rule order can be dynamically modified, based on their matching frequencies. Since the problem of optimal rule ordering with precedence constraints is NP-complete, authors proposed a heuristic approximation algorithm able to provide satisfactory optimization in reasonable time.

In [15] Fulp modeled the precedence relations among rules as a DAG and proposed a simple bubble sort-like heuristic algorithm based on admissible swaps of adjacent rules. The algorithm was then improved in [16]. In [17] Mohan *et al.* extended Fulp's algorithm by finding admissible swapping windows, i.e., rules sequences where the first and last rules can be exchanged without violating precedence constraints. All these works assume that the matching frequencies be independent of the rule position in the sequence.

The authors of [18] observed that a firewall integrity is also preserved when two dependent rules are swapped, which are associated to the same action. As the matching frequencies of the exchanged rules likely change, the authors proposed an algorithm which takes into account the impact of any exchange on the average packet processing time.

2) Rule Compression: Both [19] and [20] presented algorithms to compute a configuration equivalent to a given firewall, where the number of rules is close to the minimum. The compression algorithm in [19] is based on the transformation of the rule sequence into a decision diagram, whereas [20] changes higher dimensional target patterns into lower elements by dividing the original pattern into hyperplanes and then suitably resolving differences between two adjacent hyperplanes. Rule compression strategies are designed to be run offline so that administrators only have to manage the uncompressed rules, avoiding the quite cryptic semantic of compressed version.

It is worth observing that the solution proposed in this paper is compatible with both the ordering and compression techniques mentioned above, therefore it can be profitably adopted in conjunction with them to achieve further improvements in the overall firewall performance.

3) Rule Analysis: The scientific literature about the analysis of rules is quite large and a good survey can be found in [21]. Indeed, several techniques have been proposed to assist network security administrators in detecting policy conflicts and anomalies concerning either a single device or multiple connected firewalls. As an example, the technique proposed in [22] allows to discover anomalies such as rule shadowing, redundancy and irrelevance, operating at both intra- and inter-firewall level. Detected anomalies are not necessarily errors in policy translation, but the technique helps administrators in checking the configuration of their systems. Approaches like [22] are typically adopted as a first optimization step after the initial network configuration and, as such, are compatible with the solution introduced in this paper.

B. Inter-Firewalls Solutions

Inter-firewall solutions belong to two main classes depending on whether they rely on specific network architectures possibly with redundant h/w, e.g., parallel firewalls, or they exploit generic multi-firewall systems.

1) Parallel Architectures: Those solutions are explicitly conceived and adopted in the design phase of the network, to improve the performance by spreading the filtering load over a number of firewalls in parallel. From this point of view the technique introduced in this paper is more flexible. In fact, it can be used to reconfigure at run-time firewalls already deployed in the field, as well as help during the design phase to distribute the expected filtering load.

In [23] Fulp presented some solutions leveraging the use of parallel firewalls to distribute either packets, to reduce the filtering load of given devices, or rules in such a way that each packet is processed by several firewalls and a decision is taken by a gate which compares their resulting actions.

In [24] a similar architecture was proposed focusing on fault-tolerant implementation and algorithms for load distribution. Finally, in [25] a technique was described to distribute both rules and packets among parallel firewalls while maintaining the network security integrity. This solution first preprocesses rules to eliminate their dependencies and translates the firewall sequence to include only *accept* actions. Then rules are assigned to firewalls depending on their matching capabilities. In addition to rules processing, packet distribution is also performed, based on similar criteria, by means of meta-rules implemented in a pre-filtering stage.

Though solutions based on parallel firewalls provide undeniable benefits in terms of both performance enhancement and tolerance to heavy traffic loads, they are quite expensive and can hardly abstract from the use of additional hardware.

2) Generic Multi-Firewall Architectures: These solutions, like our proposal, are aimed at reducing the filtering load of a firewall, whose packet processing latency becomes for some reason critical, through the assistance of existing/deployable h/w and/or s/w neighbors. Basic requirements for such a goal are the following:

- The methodology should be applicable to operational networks, although it can also be used during the design phase to efficiently distribute the expected load.
- An orchestration/coordination infrastructure is needed, which must be able to detect the potentially dangerous situations and to compute and deploy the configuration changes by taking into account traffic statistics, involved firewalls and related network objects and devices.

The dynamic redistribution of rules between different FWs, in order to cope with changes in the filtered traffic profile, is a technique also adopted by other authors and, in particular, in [26] and [27], our approach differs in the way we redistribute the rules and the kind of networks we model.

In particular, to improve the behavior of an overloaded firewall, [26] and [27] move set of rules (and consequent filtering load) to FWs which are closer to the source of traffic peaks. Conversely, our approach redistributes rules to downstream firewalls at the expenses of an increase in communication bandwidth consumption. Reference scenarios are also different in the two cases: in fact [27] considers a cloud-oriented infrastructure where the instantiation of lightweight s/w FWs is possible, whereas networks targeted in this paper are classical tree-shaped architectures, such as those commonly found in industrial and automation environments or cyber-physical systems. Actually, in most of those situations the network topology is quite rigid (and cannot be changed easily): FWs are used to protect different plant areas and hierarchically organized from the outer (i.e., the perimeter Internet interface) to the inner line of defense (i.e., the shopfloor subnetwork) [2]. Connected devices have often low computation power and run proprietary s/w which does not enable the dynamic creation of additional tasks (i.e., firewalls), as this could also affect the effectiveness of the control action. By contrast the available communication resources are often underused and this makes acceptable the penalty involved in our solution in terms of increased bandwidth usage.

Preserving the overall filtering behavior is mandatory for each transformation and this has been formally proven for the approach presented in this paper. In addition, perimeter FWs have often to manage high loads in many networks, either physical or cloud-based, as there is no other device closer to the traffic source which could be able to reduce the FW filtering effort. Literature approaches based on dynamic reconfiguration of upstream firewalls require changes in either the routing paths, such as in [27], or in the filtered packets, such as in [26], to prevent the lightened FW from processing already filtered packets or alternatively to enable their faster analysis. This kind of interventions is not needed with our technique which is totally transparent to the routing mechanisms and packet structures.

It is also worth noting that both [26] and [27] apply rule reordering algorithms to each FW in order to achieve the goal of having most packets filtered by primarily checked rules. This optimization is not incompatible with our proposal though it was not considered in this paper: in principle it can be applied to our solution too, and further investigations are going to be carried out in future work.

In more details, authors of [26] adopt sound formal techniques too, with the goal of fairly equalize the workload of FWs along the critical traffic path. This roughly means that all FWs traversed by the critical stream should be equally loaded, i.e. the overall workload is homogeneously distributed among all the affected FWs. Broadly speaking, such a workload is conceptually close to the filtering load we introduced in previous sections: the cost paid to decide whether to drop or forward a packet depends on the position of the matching rule in the FW sequence, that is the number of rules unsuccessfully checked before the matching is found. However, in [26]:

- Only *accept*, *drop*/*deny* rules are considered, whereas our model includes also rule-skipping actions, which are now widely used in a number of open-source and proprietary products [39]–[43].
- The firewalls semantics is changed, since the workload distribution relies on further checks that each firewall has to perform about unused fields in IP headers that have been purposely employed to enable the proposed approach. Our technique, instead, is perfectly compatible with most existing devices and it does not affect the FW semantics.
- Both algorithms in [26] and Fig. 3 can run in polynomial time, but the two polynomial functions differ in the type and number of underlying variables.

Authors of [26] measure the performance improvement by comparing the *highest normalized workload* of the involved FWs before and after their transformation, and obtain an average workload reduction equal to 60% for the considered configurations. Such a metric can be easily compared to the inverse of Γ_{fw_1,fw_2}, plotted in Figs. 5 and 6, and experimentally evaluated in Sect. IV-C. In particular, the tenth column in Tab. I shows the gain of the (FW_1, FW_2) pair, i.e. the average gain for each firewall if the total workload is assumed to be evenly split between the two devices. Reciprocals of values in the column range from 0.53 to 0.73, corresponding to an average load reduction equal to 47% and 27% respectively, that is values not so far from 60% in [26].

Therefore we have closely comparable gains, with much less complexity and with full compatibility with existing devices.

As mentioned before, a cloudy environment is considered in [27], where performance and resilience to DoS attacks are improved by leveraging a cloud/cloudlets architecture. The firewall is assumed to be located in the cloud and, when a distributed controller (cloud security controller) detects changes in the traffic from a cloudlet, a virtual firewall is created there, and relevant rules are moved from the existing (central) firewall to the new virtual device. In particular:

- Decisions taken quickly at the system edges in [27] have the advantage of blocking unwanted/malicious traffic as soon as possible by modifying the packet routing. Our approach does not involve any change in routing, and packets traversing the newly configured firewalls to reach their destinations are filtered exactly the same way as before the rule migration.
- The integrity of the network security is claimed to be preserved in [27] (i.e., each packet is delivered or dropped in the same way before and after the transformation), but no proof is provided in that paper. Our formal approach makes such a proof easy to be obtained together with a confirmation of the applied transformation correctness.
- The computational complexity of the approach in [27] is kept manageable thanks to a pre-processing step able to eliminate the rule dependencies in the macro firewall, i.e. the relieved firewall. This is not needed in our case.

The reference scenario considered in [27] for evaluating performance and the related metric cannot be put into direct correspondence with our solution, anyway a rough comparison is possible by considering their configuration where just one auxiliary FW is instantiated to relieve the overloaded central device. Their metric, called *Packet Processing Time* (PPT) measures the time spent by a firewall to check packets by assuming that this cost is linearly proportional to the position of the matched rule in the FW sequence. In [27] the authors provide the PPT absolute values for the critical firewall before and after their proposed transformation. Despite the differences of the considered scenarios with respect to our solution, it is easy to verify that, in this condition, their improvement ratios fall between 2 and 3. These results are rather comparable and not far from the measured l_1/l_2 values shown in the seventh column of Tab. I.

VI. Conclusions

The (re)distribution of filtering rules between different firewalls located in the same network is a technical challenge which is key not only in the design phase of a new system, but also during the real-time operation and management of the network itself. Of course, this could not be possible without a strict coordination of the firewall activities, but solutions are now offered by well-assessed paradigms such as SDN and NFV, providing the levels of flexibility and supervision needed for this purpose.

In this paper a novel technique has been presented, which is based on the redistribution of the filtering rules between cascaded firewalls, in order to reduce the packet processing overhead in overload conditions, such as temporary traffic peaks or DoS attacks. The proposed methodology is based on a simple formal model which allows the modeling of the firewall capabilities without introducing unnecessary details for the purpose of redistributing the packet filtering load between different devices.

Our solution requires neither changes to the routing infrastructure nor modification of packets, such as usage of additional bit fields, and is highly compatible with devices and network equipment typically found in factory and automation infrastructures.

A transformation algorithm has been introduced to move rules between different FWs, its correctness and ability to preserve the security of the network have been formally proven and verified through simulation. The REDIAL algorithm can be usefully adopted in either the management of an existing network, i.e., for fine-tuning operations, or the design and planning phases to estimate the performance of a system before its actual implementation and deployment. It has polynomial complexity and can be run to determine new firewall configurations in real-time with moderate computing resources.

We also computed theoretical performance bounds for the proposed technique in order to quantitatively predict the advantages in terms of packet processing rates that can be obtained by applying the FW transformation to the network of interest.

Simulation results obtained in a number of experiments by varying the traffic and firewall characteristics confirm the validity of the proposed approach and a very good accordance with the theoretical model.

Measures collected, by deploying the proposed solution in a purposely developed test-bed, show that the expected performance gain determined both theoretically and through simulation is close to the actual improvements achievable in realistic conditions.

Finally, our solution is not incompatible with other intra-firewall optimization techniques such as rule ordering, compression and analysis and can be used in conjunction with them in order to achieve highly effective cybersecurity schemes.

Acknowledgment

Computational resources were provided by HPC@POLITO, a project of Academic Computing within the Department of Control and Computer Engineering at the Politecnico di Torino (http://hpc.polito.it), Torino, Italy. The authors are grateful to Dr. Manuel Cheminod for his help in setting up the experimental test-bed and running the experiments.

References

[1] K. Stouffer, V. Pillitteri, S. Lightman, M. Abrams, and A. Hanh, “Guide to industrial control systems (ICS) security,” Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. NIST SP 800-82, Revision 2, May 2015, doi: 10.6028/NIST.SP.800-82r2.

[2] M. Cheminod, L. Durante, and A. Valenzano, “Review of security issues in industrial networks,” *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 277–293, Feb. 2013.

[3] M. Cheminod, L. Durante, L. Seno, and A. Valenzano, “Performance evaluation and modeling of an industrial application-layer firewall,” *IEEE Trans. Ind. Informat.*, vol. 14, no. 5, pp. 2159–2170, May 2018.

[4] C. Zunino, A. Valenzano, R. Obermaisser, and S. Petersen, “Factory communications at the dawn of the fourth industrial revolution,” *Comput. Standards Interfaces*, vol. 71, Aug. 2020, Art. no. 103433.

[5] K. Scarfone and P. Hofman, “Guidelines on firewalls and firewall policy,” Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. NIST SP 800-41, Revision 1, Sep. 2009, doi: 10.6028/NIST.SP.800-41r1.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[7] R. Mahmud, R. Kotagiri, and R. Buyya, *Fog Computing: A Taxonomy, Survey and Future Directions* (Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives). Singapore: Springer, 2018, pp. 103–130.

[8] P. P. Ray, “A survey on Internet of Things architectures,” *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 30, no. 3, pp. 291–319, 2018.

[9] C. C. Sobin, “A survey on architecture, protocols and challenges in IoT,” *Wireless Pers. Commun.*, vol. 112, no. 3, pp. 1383–1429, Jun. 2020.

[10] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, “The industrial Internet of Things (IIoT): An analysis framework,” *Comput. Ind.*, vol. 101, pp. 1–12, Oct. 2018.

[11] H. Xu, W. Yu, D. Griffith, and N. Golmie, “A survey on industrial Internet of Things: A cyber-physical systems perspective,” *IEEE Access*, vol. 6, pp. 78238–78259, Dec. 2018.

[12] Y. Lu, “Industry 4.0: A survey on technologies, applications and open research issues,” *J. Ind. Inf. Integr.*, vol. 6, pp. 1–10, Jun. 2017.

[13] L. D. Xu, E. L. Xu, and L. Li, “Industry 4.0: State of the art and future trends,” *Int. J. Prod. Res.*, vol. 56, no. 8, pp. 2941–2962, Apr. 2018.

[14] H. Hamed and E. Al-Shaer, “Dynamic rule-ordering optimization for high-speed firewall filtering,” in *Proc. ACM Symp. Inf., Comput. Commun. Secur. (ASIACCS)*, 2006, pp. 332–342.

[15] E. W. Fulp, “Optimization of network firewall policies using directed acyclical graphs,” in *Proc. IEEE Internet Manage. Conf. (IM)*, Jan. 2005, pp. 1–4.

[16] A. Tapdiya and E. W. Fulp, “Towards optimal firewall rule ordering utilizing directed acyclical graphs,” in *Proc. 18th Int. Conf. Comput. Commun. Netw. (ICCN)*, Aug. 2009, pp. 1–6.

[17] R. Mohan, A. Yazidi, B. Feng, and J. Oommen, “On optimizing firewall performance in dynamic networks by invoking a novelswapping window-based paradigm,” *Int. J. Commun. Syst.*, vol. 31, no. 15, p. e3773, Jul. 2018.

[18] T. Harada, K. Tanaka, and K. Mikawa, “A heuristic algorithm for relaxed optimal rule ordering problem,” in *Proc. 2nd Cyber Secur. Netw. Conf. (CSNet)*, Oct. 2018, pp. 1–8.

[19] A. X. Liu, E. Torng, and C. R. Meiners, “Firewall compressor: An algorithm for minimizing firewall policies,” in *Proc. 27th Conf. Comput. Commun. (INFOCOM)*, Apr. 2008, pp. 176–180.

[20] J. Daly, A. X. Liu, and E. Torng, “A difference resolution approach to compressing access control lists,” *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 610–623, Feb. 2016.

[21] A. A. Jabal *et al.*, “Methods and tools for policy analysis,” *ACM Comput. Surv.*, vol. 51, no. 6, pp. 1–35, Feb. 2019.

[22] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, “Conflict classification and analysis of distributed firewall policies,” *IEEE J. Sel. Areas Commun.*, vol. 23, no. 10, pp. 2069–2084, Oct. 2005.

[23] E. W. Fulp, “Parallel firewall designs for high-speed networks,” in *Proc. IEEE 25th Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2006, pp. 1–4.
[24] P. Zhichao, C. Daiwu, and H. Wenhua, “A load-balancing and state-sharing algorithm for fault-tolerant firewall cluster,” in *Proc. 4th Int. Conf. Inf. Sci. Control Eng. (ICISCE)*, Jul. 2017, pp. 34–37.
[25] T. E. Hadjadj, R. Tebourbi, A. Bouhoula, and R. Ksantini, “Optimization of parallel firewalls filtering rules,” in *Proc. Int. Conf. Softw., Telecommun. Comput. Netw. (SoftCOM)*, Sep. 2019, pp. 1–6.
[26] G. Yan, S. Chen, and S. Eidenbenz, “Dynamic balancing of packet filtering workloads on distributed firewalls,” in *Proc. 16th Int. Workshop Qual. Service (WQOS)*, Jun. 2008, pp. 209–218.
[27] S. Bagheri and A. Shameli-Sendi, “Dynamic firewall decomposition and composition in the cloud,” *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3526–3539, Apr. 2020.
[28] M. Cheminod, L. Durante, A. Valenzano, and C. Zunino, “Performance impact of commercial industrial firewalls on networked control systems,” in *Proc. IEEE 21st Int. Conf. Emerg. Technol. Factory Automat. (ETFA)*, Sep. 2016, pp. 1–8.
[29] CISCO. *Defense Orchestrator*. Accessed: Feb. 5, 2021. [Online]. Available: https://docs.defenseorchestrator.com/
[30] Fortinet. *FortiManager*. Accessed: Feb. 5, 2021. [Online]. Available: https://www.fortinet.com/products/management/fortimanager/models-specs/
[31] S. Kim, S. Yoon, J. Narantuya, and H. Lim, “Secure collecting, optimizing, and deploying of firewall rules in software-defined networks,” *IEEE Access*, vol. 8, pp. 15166–15177, Jan. 2020.
[32] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, “FLOWGUARD: Building robust firewalls for software-defined networks,” in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, Aug. 2014, pp. 97–102.
[33] J. N. Bakker, I. Welch, and W. K. G. Seah, “Network-wide virtual firewall using SDN/OpenFlow,” in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2016, pp. 62–68.
[34] J. Deng *et al.*, “VNGuard: An NFV/SDN combination framework for provisioning and managing virtual firewalls,” in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2015, pp. 107–114.
[35] M. Cheminod, L. Durante, M. Maggiora, A. Valenzano, and C. Zunino, “Performance of firewalls for industrial application,” in *Proc. 4th Int. Symp. ICS SCADA Cyber Secur. Res. (ICS-CSR)*, Aug. 2016, pp. 42–52.
[36] *Intel Ethernet Server Adapter I350*. Accessed: Feb. 5, 2021. [Online]. Available: https://www.intel.com/content/www/us/en/products/docs/network-io/ethernet/10-25-40-gigabit-adapters/ethernet-i350-server-adapter-brief.html
[37] *Keysight Copper TAP TP-CU3*. Accessed: Feb. 5, 2021. [Online]. Available: https://www.keysight.com/it/en/products/network-visibility/network-taps/copper-taps.html
[38] *Belden MACH104-20TX-F-L3P-Managed 24-Port Full Gigabit 19 Switch With L3*. Accessed: Feb. 5, 2021. [Online]. Available: https://catalog.belden.com/index.cfm?event=pd&p=PF_942003002
[39] *The Netfilter Project*. Accessed: Feb. 5, 2021. [Online]. Available: https://www.netfilter.org/
[40] *The Nftables Project*. Accessed: Feb. 5, 2021. [Online]. Available: https://wiki.nftables.org/
[41] *The IPFW Firewall*. Accessed: Feb. 5, 2021. [Online]. Available: https://docs.freebsd.org/en/books/handbook/firewalls/#firewalls-ipfw
[42] *Packet Filter (PF)*. Accessed: Feb. 5, 2021. [Online]. Available: https://www.openbsd.org/faq/pf/
[43] *Next Generation FireWall*. Accessed: Feb. 5, 2021. [Online]. Available: https://www.forcepoint.com/product/ngfw-next-generation-firewall

Luca Durante received the Ph.D. degree in computer engineering from the Politecnico di Torino in 1996. He is currently a Senior Researcher with the Institute of Electronics, Information Engineering and Telecommunications, National Research Council of Italy (CNR-IEIIT), Torino, Italy. He has coauthored about 70 scientific journal and conference papers, and he has been the Scientific Coordinator of the CNR Team in European Projects. He has also served as the General Co-Chair, Program Co-Chair, and TPC member for several international conferences and referee for international journals. His current research interests include access control policies for security and security of cyber-physical systems.

Lucia Seno (Member, IEEE) received the B.S. and M.S. degrees in automation engineering and the Ph.D. degree in information engineering from the University of Padova, Padova, Italy, in 2004, 2007, and 2011, respectively.

Since then, she has been with the Institute of Electronics, Information Engineering, and Telecommunications, National Research Council of Italy in Padova, and subsequently, in Torino, Italy, where she is currently a Researcher. Her research interests include industrial communication systems and technologies, wireless networks and wireless sensor networks for real-time and safety-critical control applications, formal verification of systems security and communication protocols, and model checking.

Adriano Valenzano (Senior Member, IEEE) received the Laurea degree *(magna cum laude)* in electronic engineering from the Politecnico di Torino, Torino, Italy, in 1980.

He is currently a Director of Research with the National Research Council of Italy (CNR). He is currently with the Institute of Electronics, Computer and Telecommunication Engineering (IEIIT), Torino, where he is responsible for research concerning distributed computer systems, local area networks, and communication protocols. He has coauthored approximately 200 refereed journal and conference papers in the area of computer engineering.

Dr. Valenzano is the recipient of the 2013 IEEE IES and ABB Lifetime Contribution to Factory Automation Award. He was also awarded for the best paper published in the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS during 2016, and received the Best Paper Awards for the papers presented at the 5th, 8th, 13th, 15th, and 16th IEEE International Conference on Factory Communication Systems. He has served as a Technical Referee for several international journals and conferences, also taking part in the program committees of international events of primary importance. Since 2007, he has been serving as an Associate Editor for the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS.