Hindawi Publishing Corporation
Journal of Computer Networks and Communications
Volume 2015, Article ID 872326, 17 pages
http://dx.doi.org/10.1155/2015/872326

Research Article

Adaptive Conflict-Free Optimization of Rule Sets for Network Security Packet Filtering Devices

Andrea Baiocchi,[1] Gianluca Maiolini,[2] Annachiara Mingo,[3] and Daniele Goretti[4]

[1]*Department of Information Engineering, Electronics and Telecommunications (DIET), University of Roma "Sapienza", Via Eudossiana 18, 00184 Rome, Italy*
[2]*Ipanema Technologies, Via Roberto Lepetit 8/10, 20124 Milan, Italy*
[3]*Digi International GmbH, Lise-Meitner-Straße 9, 85737 Ismaning, Germany*
[4]*Altran Italia S.p.A., Via Tiburtina 1232, 00131 Rome, Italy*

Correspondence should be addressed to Andrea Baiocchi; andrea.baiocchi@uniroma1.it

Received 1 June 2014; Revised 17 December 2014; Accepted 22 December 2014

Academic Editor: Tin-Yu Wu

Packet filtering and processing rules management in firewalls and security gateways has become commonplace in increasingly complex networks. On one side there is a need to maintain the logic of high level policies, which requires administrators to implement and update a large amount of filtering rules while keeping them conflict-free, that is, avoiding security inconsistencies. On the other side, traffic adaptive optimization of large rule lists is useful for general purpose computers used as filtering devices, without specific designed hardware, to face growing link speeds and to harden filtering devices against DoS and DDoS attacks. Our work joins the two issues in an innovative way and defines a traffic adaptive algorithm to find conflict-free optimized rule sets, by relying on information gathered with traffic logs. The proposed approach suits current technology architectures and exploits available features, like traffic log databases, to minimize the impact of ACO development on the packet filtering devices. We demonstrate the benefit entailed by the proposed algorithm through measurements on a test bed made up of real-life, commercial packet filtering devices.

1. Introduction

A key challenge of secure systems is the management of security policies, from high level ones down to the platform specific implementation. Security policies define constraints, limitations, and authorization on data handling and communications. The growth of communication links speed brings forward a need for improved performance of packet filtering devices, such as firewalls and secure Virtual Private Networks (S-VPN) gateways. To improve performance while maintaining consistency, network security policies should be tailored according to the network traffic. We address specifically computer based packet filtering devices that do not use hardware specialized filters (e.g., based on FPGAs) and refer to that vastly widespread sequential rule list model, which accounts for most common, computer-based filtering devices currently deployed.

The process of inspecting incoming packets and looking up the policy rule set for a match often results in CPU overload and packet delay or even loss. As a matter of fact, rule lists do not exceed few hundreds active rules in well-maintained, operational packet filtering devices. Packets that match high rank rules require a small computation time compared to those that require scanning the whole rule set. The processing load per packet becomes increasingly concerning as the input line speed increases and as packet filtering functions are assigned to a larger number of inexpensive, relatively simple devices. Having packets matching high rank rules is not so unlikely; for example, typically undesired or unpredicted traffic is essentially dealt with by the "deny all" rule.

In this paper, we pursue saving of CPU power by shaping the rule set onto the network traffic impacting the device. The idea is to give high priority to rules intercepting a large fraction of current traffic. Algorithms aiming at packet

filter processing time improvements are presented in [1–6]. The nontriviality of the optimization procedures is due to dependencies among rules, which puts constraints on rule reordering. Disregarding such dependencies can introduce inconsistency of policies implemented by the rule set of the devices. As reported in a number of works [7–11], conflicts among rules can cause holes in security, which are often hard to detect.

We develop an algorithm to solve the rule set optimization problem, under the constraint that the reordered rule set be conflict-free. Leveraging on this approach, already proposed, for example, in [5], we extend the optimization algorithm with the extraction of new rules from the "deny all" rule, in order to improve packet processing time further by capturing undesired packet flows that do not match any of the existing rules. The new rules are inserted in the rule set so as to maintain the optimization of the processing load with respect to the current traffic mix. The overall optimization procedure is named Adaptive Conflict-Free Optimization (ACO). Our test results prove that the extraction of rules from the "deny all" rule, as done in ACO, can improve CPU performance of packet filtering devices and it can reduce the impact of DoS (Denial of Service) and DDoS (Distributed Denial of Service) attacks.

We outline an adaptive procedure to automatically launch ACO according to traffic profile measured at device interfaces, aiming at striking a balance between device configuration updates and obtainable performance gains in a time varying environment, where traffic mix changes over time. Information on the network traffic mix is retrieved from log files collected by packet filtering devices. Using log files directly from the packet filtering device allows us to define an adaption algorithm that can be used for different kind of filtering, that is, whatever the fields exploited by rules are (e.g., header based or based on application level payload strings).

Our aim is to show how relatively simple means can gain a performance improvement without deeply affecting hardware and software of currently deployed devices, especially in the access networks, where their number is large and they are based on relative cheap, off-the-shelf machines.

The paper is organized as follows. In Section 2 we describe related works. In Section 3 we introduce the operational scenario and the software tools we have realized to run ACO. A detailed description of ACO algorithm is provided in Section 4. Section 5 outlines the algorithm that launches the ACO adaptively, according to the measured traffic mix. In Section 6 we describe experimental results based on a laboratory test-bed aiming at measuring ACO performance improvement and effectiveness against DoS attacks. Finally, we give some concluding remarks in Section 7.

2. Related Work

In [12] the Policy Core Information Model (PCIM) is described as an object-oriented model for representing policy information as extensions to the Common Information Model (CIM) activity within the Distributed Management Task Force (DMTF: http://www.dmtf.org/). The definition of policy and policy rule presented in PCIM and its extension shown in RFC-3198 [13] gave to Basile and Lioy [14] the starting point to refine these concepts in a way useful for a formal approach. Hari et al. [7] aim at detecting if firewall rules are correlated to each other, while in [8, 9] a set of techniques and algorithms are defined to detect all policy conflicts. Along this line, [10] and [11] provide an automatic conflict resolution algorithm for a single firewall and a tuning algorithm for multiple cooperating firewalls, respectively.

In parallel, great emphasis has been placed on how to optimize packet filtering devices performance. The recent review in [15] offers a systematic comparison of traffic-aware approaches to rule-based traffic filtering in security devices. In [16] a simple algorithm based on rule reordering is presented. This work describes rule dependencies using Directed Acyclic Graphs (DAGs), yet it does not provide a methodology to build the DAG of a given device. In addition the proposed algorithm is unfeasible in a real environments with large rule sets and complex graphs. Framework and methodologies to inspect and analyze both multidimensional firewall rules and traffic logs information are proposed in [1–3]. In [1, 2] the optimization tool uses current traffic characteristics to define rule set ordering so as to minimize the operational cost of the firewall. Four schemes are used to achieve this goal (hot caching, total reordering, default proxy, and online adaptation). In [3] an adaptive firewall optimization framework, named OPTWALL, is proposed; it is built to reflect the current traffic pattern into rule sets. A limit of [1–3] is that it is not defined when the update process must be started and the weight parameters used in the rule size estimation. The approach proposed in [4] optimizes the performance by rule reordering, but how to create the necessary statistics for rule weight estimation as well as how to find dependency relations between rules is not defined. In [5] an algorithm to optimize firewall performance is presented; it orders the rules according to their weights and considers two factors to determine the weight of a rule: rule frequency and recency which reflect the number and time of rule matching, respectively. They present two types of update: performance-based triggered update and time-based periodic update. We adopt a similar approach, also taking into account for the further optimization brought about by breaking up the default "deny-all" rule. Reference [6] presents a process of managing firewall policy rules, consisting of anomaly detection, generalization, and policy update using Association Rule Mining and frequency-based techniques. However, a complex distributed network with multiple firewalls and log acquisition are not contemplated. TCAM based fast packet classification is proposed in [17]. However, TCAM are expensive and power hungry, as pointed out, for example, in [18]. Efficient packet classification by means of an especially designed software is tackled in [19]. In [18], after a wide review of many alternatives, Lim et al. propose and analyse the Boundary Cutting algorithm. It leads to a decision tree data structure that can be optimised to yield good search complexity even in very big rule lists (in the order of 100000 rules). A heuristic approach is explored in [20], by looking for a compromise between memory efficient trie data structures and search efficient decision trees. Detection

of specific packets is considered in [21], where a randomised algorithm is considered: the emphasis here is placed on isolating specifically targeted packets from the mass of the wire traffic. Even though mere search performance can be quite improved by decision tree, still complexity, power consumption, and cost often call for simpler realization of packet filtering devices. So, adapting the rule list to the current traffic load remains a valid concept. Following that concept, an approach similar to ours, yet based on a more complex algorithm than the one we have developed, is defined in [22]. In [23, 24] different traffic-aware packet classification algorithms are defined, without considering specifically the traffic-adaptive optimization obtained by extracting detailed rules from the "deny all" rule. The rejection of massive undesired traffic is addressed in [25]. Their approach can be seen as complementary to the one here proposed, based on the extraction of new rules from the "deny all" rule.

A third relevant and correlated issue is about the impact of the rule extraction from the deny all string. The few works on this topic [1, 4] do not demonstrate if and in which cases this action benefits on CPU processing time. Moreover, those works do not detail how many rules should be extracted and according to which priority order. We give an extraction algorithm coupled with rule set optimization and demonstrate it can help relieving the effect of Denial of Service (DoS) attacks on the packet filtering devices. DoS attacks attempt to exhaust or disable access to resources at the victim. These resources are either network bandwidth, computing power, or operating system data structures. In flooding attacks, one or more attackers send streams of packets aimed at overwhelming link bandwidth or computing resources at the victim [26]. This type of attack, defined in [27], can be really dangerous because it can be performed also by using many unaware sources of attack (Distributed DoS), so reaching huge diffusion and volume, as shown in [28], where a three-week analysis of a network is reported that found more than 12000 DoS attacks. In particular, we focused our attention on a flooding attack towards a firewall, aiming at making the packet filtering device collapse by means of a huge quantity of messages matching "deny all" rule.

Current packet filtering technologies exploit traffic adaptive mechanisms, as take-in access list in cache [29]. In particular, the device stores a hash table whose entries match active packet flows and point at the corresponding rule/action of the rule set (*cache association*). This allows scanning the rule set only for the first packet of each active flow. Despite this method being adaptive to network traffic, its efficiency decreases when the size of the hash table grows. Moreover, this approach is ineffective with a large number of different undesired packet flows.

Finally, we give just a hint to different research directions on packet filtering devices. High speed packet filtering by means of specialized and optimized hardware is a prolific topic; for example, some recent works address the use of FPGAs (e.g., [30–33]). These works focus on optimized hardware design or matching rule searching techniques that can be conveniently implemented with FPGAs. Instead, in this work we assume a general purpose computer server is used to run the filtering machine, which is typical of access networks devices. Another approach focuses on defining an efficient compiler to produce optimized implementation of a high level policy list, to minimize match search complexity (e.g., see [34, 35]). These works focus on optimization of the code implementing the filtering machine for the given list of rules, while our approach aims at optimally adapting the sorting of the rule list to the current analyzed traffic mix. These can be seen as complementary points of view.

3. System Architecture

3.1. Definitions and Notation. We assume that security policies are translated into an ordered list of predicates of the form: $C \rightarrow A$, where C is a condition and A is an action. We refer to predicates implementing security policies as *rules*. For security gateways and packet filtering devices, actions A that can be carried out on a packet are *allow* or *deny* (In IPSec gateways a third possible action is *process* for packets belonging to an activated security association needing to be encrypted and/or protected for authentication and integrity check.). The condition of a rule is obtained as the logical AND of a number of conditions of the type: "selector value from packet header/payload belongs to a given interval or set/matches the given string." For example, classic implementation of network level packet filtering devices considers five selectors:

(1) PT = *protocol_type*, whose values can be represented by eight bit integers, that is, range between 0 and 2^8-1;

(2) SA = *source_ip_address* and DA = *destination_ip_address*, whose values can be represented in dotted decimal notation and correspond to integers ranging from 0 up to $2^{32} - 1$ (for IPv4);

(3) SP = *source_port* and DP = *destination_port*, whose values can be represented by sixteen bit integers, that is, range between 0 and $2^{16} - 1$.

A condition is specified by giving an interval of values for each selector; that is, a condition can be viewed as an interval contained in the five-dimensional, finite lattice space S^5 defined by

$$\begin{aligned} S^5 = \big\{ & 0 \le \mathrm{SA} \le 2^{32} - 1;\ 0 \le \mathrm{DA} \le 2^{32} - 1; \\ & 0 \le \mathrm{SP} \le 2^{16} - 1;\ 0 \le \mathrm{DP} \le 2^{16} - 1; \\ & 0 \le \mathrm{PT} \le 2^{8} - 1 \big\}. \end{aligned} \tag{1}$$

Different selectors could be considered, possibly involving header fields belonging to other layers than network one, for example, application layer, or using strings taken from packet payload. For example, a URL can be used in the rule condition. The basic structure of the list as a sequence of rules does not change though. In the end, the predicates reduce to text strings or to numeric intervals. The selected fields of each packet are checked against the predicates to verify whether they correspond to the string value or are comprised within the interval range.

Given a rule set organized as an ordered list, each packet delivered to the packet filtering device interfaces is checked against each rule, following the rule ordering, until the *first* matching rule is found. Then, the action of the matching rule is applied. The last rule, R_{N+1}, is usually a "deny all," that is, a rule with wild-cards for each condition field. The "deny all" discards any packet that has not matched any previous rule, so it implements the principle that anything which is not explicitly allowed must be denied. We assume there is always a "deny all" at the bottom of the rule list.

The processing cost per packet is proportional to the depth of the matching rule. Hence, it can be reduced by reordering the rules according to the fraction of the input load that matches each rule, under the constraint of maintaining the dependencies among rules. The adaptation algorithm of a tagged device is triggered only when the analysis of the overall hit ratios of rules of that device points out that a significant shift of the aggregated traffic mix through the tagged device has taken place. The traffic mix is monitored through the logs produced by the device itself, as detailed in the ensuing subsection.

3.2. Networking Scenario. The considered scenario is made up of packet filtering security devices deployed in a managed network. Network Management Systems (NMS) allow administrators to handle device configurations (rule lists) and to monitor packets flowing through devices using log messages collected and stored by the packet filtering device.

The overall architecture of the automated and adaptive policy management system that we have built up is depicted in Figure 1. The complete system comprises a policy conflict resolution tool, a log management infrastructure, and a tool that, based on log messages collected from all devices in the network, estimates rule matching ratios and triggers automatically and adaptively the rule set optimization process based on traffic statistics. The focus of this paper is on the optimization and adaptation part of the entire project.

All packet filtering devices, such as firewalls and security gateways, are set up to collect and send a log message reporting on packets they allow or deny as a normal part of their operations. We exploit this feature for ACO. The analysis of log messages allows us to figure out

(i) real time traffic profile without using further devices such as network agents;

(ii) how many rules are working and how many packets match with each rule.

A monitoring infrastructure is developed in order to collect and store log information into a log database (LogDB). In our testbed logs collected from devices are sent by using the "syslog" standard [36, 37]. Any other format could be used as well, provided it is "spoken" by both the device and the LogDB host. Figure 2 shows example data stored in LogDB. In particular, consider the following.

(i) *IP address* is retrieved from "syslog" packet. It identifies a device interface on the network.

(ii) *Device type* specifies rule list type; device could be configured with both FW and IPSec access list (this is an optional field).

(iii) *Rule rank* is the offset of the rule reported by the log with respect to the top of the list that the rule belongs to.

(iv) *Count* is number of packets that match that rule.

The optimization tool box in Figure 1 contains ACO algorithm. It retrieves the IP addresses of device interfaces to the networks and the device rule set from the DCDB. For each device ACO retrieves rule hit numbers from LogDB. Then it calculates rule weights and hence rule costs. These are the input parameters to the optimization algorithm (see Section 4).

Log centralization is the typical architecture used in current corporate and telcos networks. Our architecture aims at showing how to exploit log data collection of the NMS *also* to improve efficiency of packet filtering devices. Log reporting and updating of rule list are normally implemented functions and a LogDB is available in most networks independently of ACO. ACO exploits those functions for its own purposes, namely, to enhance packet filtering efficiency and harden them against DoS.

Packet filtering devices of the managed network are monitored and the ACO algorithm is started when at least one of the following events occurs:

(i) rule set is modified by the administrator (such as rule insertion, modification, or removal);

(ii) network traffic changes, that is, a new flow starts, or an existing flow varies its bit rate or terminates.

The first criterion is motivated mainly to check policy consistency and the second one to optimize performance adapting to traffic. We outline an algorithm for ACO automation in Section 5 specifically for this second situation. That is the part referred to by "Intelligent Decision Support System" in Figure 1.

4. Adaptive Conflict-Free Optimization (ACO) Algorithm Description

Let $\mathbf{R} = [R_1, \ldots, R_N, R_{N+1}]$ be the ordered, conflict-free rule list, provided as input to ACO; N is the number of rules, besides the last rule, R_{N+1}, which is assumed to be "deny all." ACO aims at minimizing packet processing times, under the constraint of maintaining a conflict-free rule list. For a detailed discussion and formalization of security policy conflicts in a rule list see [7, 8, 10]. It suffices to say that, for a conflict-free list, any couple of rules in the list must be either disjoint or in an inclusive matching relation. Rules R_i and R_j are *disjoint* if no packet can match both of them; relative positions of R_i and R_j in the list are unconstrained. Rule R_i is *inclusive matching* to R_j, denoted as $R_i \subset R_j$, if any packet matching R_i also matches R_j but the converse does not hold; moreover, actions associated with R_i and R_j must be different. For the list to be conflict-free, rule R_i must precede rule R_j (*more specific rule first*). The relevant point for ACO

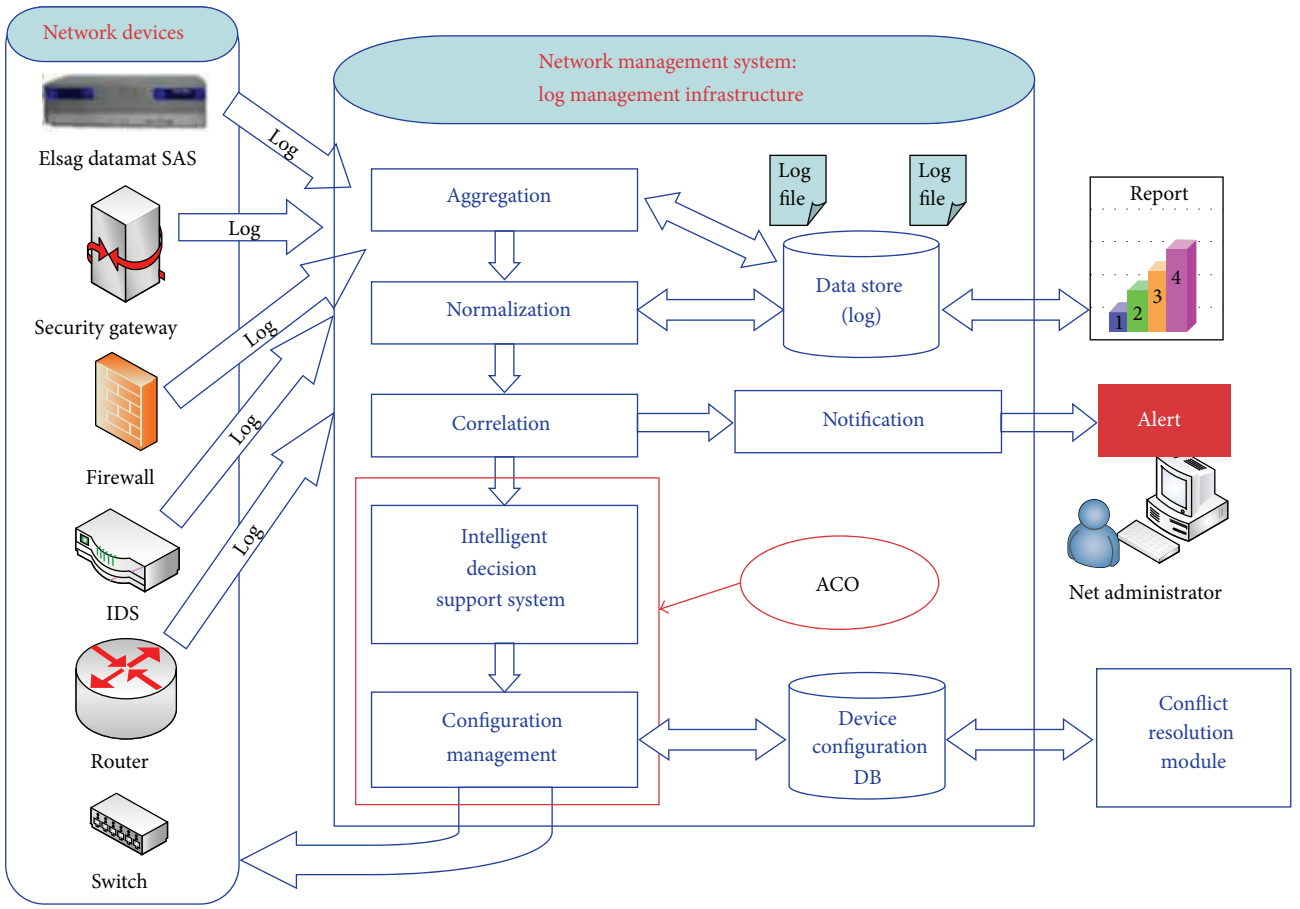
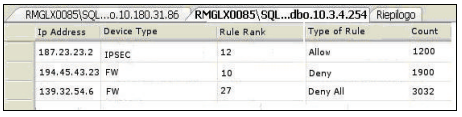


FIGURE 1: Overall architecture of the managed network system using the Adaptive Conflict-Free Optimization (ACO) module.



Ip Address	Device Type	Rule Rank	Type of Rule	Count
187.23.23.2	IPSEC	12	Allow	1200
194.45.43.23	FW	10	Deny	1900
139.32.54.6	FW	27	Deny All	3032

FIGURE 2: LogDB table example.

is that whenever two rules, say R_i and R_j, have a nondisjoint domain, that is, there exists at least one packet that matches both of them, those two rules are said to be dependent and their ordering must be preserved as given in the input rule list.

The optimization process defines a new rule list $\mathbf{R}^*$, which includes rules R_i, $i = 1, \ldots, N$, possibly reordered, and R_{N+1} ("deny all") as the last rule. Further optimization is discussed in Sections 4.1 and 4.2, by merging into the rule list also rules extracted from R_{N+1}. The optimized list $\mathbf{R}^*$ must be equivalent to $\mathbf{R}$ under the point of view of security policy implementation. Formally, for each given packet P entering the device interfaces, the action performed by the device under $\mathbf{R}$ and $\mathbf{R}^*$ must be the same.

In the following, the subscript of rules refers to their rank in the original rule list. Let y_i denote the rank of R_i in the (possibly reordered) list, $i = 1, \ldots, N+1$ (Since the "deny all" is always the last rule, it is $y_{N+!} = N + 1$; moreover, in the original rule list it is $y_i = i$.). The rank y_i is proportional to the processing cost of matching R_i; that is, for every packet matching R_i, y_i tests are required to check all rules until R_i is hit. The weight of R_i is $w_i = n_i/n$, where n_i is the number of packets hitting R_i and n is the overall number of packets received by the considered packet filtering device. The quantities n and n_i, $i = 1, \ldots, N$, are obtained by collecting the device logs over an observation time interval. The discussion of how to adapt the weights w_i over time is given in Section 5.

The cost of R_i is therefore $C_i \equiv w_i \cdot y_i$. The overall cost $C_{\mathbf{R}}(\mathbf{y})$ of the list $\mathbf{R}$ for a given rule ranking $\mathbf{y} = [y_1 \cdots y_N]$ (Any feasible ranking $\mathbf{y}$ is a permutation of the integer set $[1, 2, \ldots, N]$.) is

$$C_{\mathbf{R}}(\mathbf{y}) = C_1 + C_2 + \cdots + C_N + C_{N+1} = \sum_{i=1}^{N} w_i y_i + (N+1)\, y_{N+1}. \quad (2)$$

ACO output is a rule set $\mathbf{R}^*$ that minimizes the packet processing cost:

$$C^*_{\mathbf{R}} = \min_{\mathbf{y}} C_{\mathbf{R}}(\mathbf{y}), \tag{3}$$

under the constraint that the reordered list $\mathbf{R}^*$ be conflict-free and equivalent to $\mathbf{R}$; that is, if R_i and R_j $(i < j)$ are dependent, it must be $y_i < y_j$.

We can state the constraint in a way useful to the optimization algorithm by resorting to a Pseudo-Tree data structure describing the relationships among the rules, referred to as Device Pseudo-Tree (DPT) associated with the given rule list. An implicit definition of the DPT goes as follows: rule R_i is a child of rule R_j if and only if $R_i \subset R_j$ and there does not exist any rule R_k such that $R_i \subset R_k \subset R_j$ for $i \neq j \neq k$. Rules belonging to a conflict-free rule list, apart from the "deny all" rule, can be arranged in separate trees (possibly a single one) making up the DPT [10]. In each tree of the DPT there is a root node which represents a rule that includes all the rules in the tree and there are one or more leaves which represent the most specific rules in the tree. Given the DPT associated with $\mathbf{R}$, the constraint is checked by just requiring that no rule be assigned a rank smaller than its child rule(s); that is, scanning the list from top to bottom we must find any parent rule after its own descendant rules (i.e., the rules of the subtree rooted at the considered rule). Obviously, rules associated with disjoint subtrees of the DPT can be placed in any relative order.

The detailed steps of ACO algorithm are described in Appendix A. A full blown example of the procedure is developed in Appendix B.

4.1. Extracting Rules from Deny All. If a high rate undesired flow matches the "deny all" rule, it can be convenient to extract a specific rule for that flow and place it at the optimum rank in the rule list. Extracted rules are always disjoint from all others in the rule set, so they do not cause additional conflicts and can be placed anywhere in the rule list. However, the inclusion of extracted rules does not necessarily improve performance from processing load point of view.

In this section we define an algorithm for rule extraction from the "deny all" rule. It starts by identifying the minimum set of rules that covers the space of the denied traffic. As outlined in Section 1, the condition C of a rule $R : C \rightarrow A$ corresponds to the interval of the five-dimensional lattice S^5 described by the selector values specified in the rule condition C. We denote the interval associated with rule R_j, with $\mathcal{I}(R_j)$.

Let $\mathcal{R}$ be the set of indices of rules that are roots of the trees forming the DPT. The only rule more general than any R_i, $i \in \mathcal{R}$, is the "deny all" rule. So, for the nonredundancy of the rule list, the action associated with R_i, $i \in \mathcal{R}$, is necessarily *allow*. Then, the subspace $\mathcal{A} \subset S^5$ comprising all allowed flows is given by

$$\mathcal{A} = \bigcup_{i \in \mathcal{R}} \mathcal{I}(R_i). \tag{4}$$

Let us define $\mathcal{D}$ as the complementary space of $\mathcal{A}$ in S^5; namely, $\mathcal{D} = S^5 \setminus \mathcal{A}$. We are interested in the minimum partition of $\mathcal{D}$ into intervals; that is,

$$\mathcal{D} = \bigcup_{k=1}^{M} \mathcal{I}(R_{e,k}), \tag{5}$$

where the intervals $\mathcal{I}(R_{e,k})$ are disjoint. This partition is not unique and can be obtained efficiently, for example, by using the same techniques as in ARC/PARC (Adaptive Resolution Classifier/Pruning ARC) min-max neurofuzzy classifiers [38].

Once the intervals $\mathcal{I}(R_{e,k})$ of the partition of $\mathcal{D}$ are found, we are given the list $\mathbf{L}_e$ of extractable rules, $\mathbf{L}_e = [R_{e,1}, \ldots, R_{e,M}]$, to insert in the optimized rule set $\mathbf{R}^*$ in order to achieve a reduction of device processing load. Only those rules from $\mathbf{L}_e$ that lead to a significant processing effort saving are included into $\mathbf{R}^*$. This depends on the weight $w_{e,k}$ of $R_{e,k}$, that is, the fraction of packets matching $R_{e,k}$ during the observation interval.

For each $R_{e,k}$ in $\mathbf{L}_e$, rule weight $w_{e,k}$ in the observed time interval T can be computed as $w_{e,k} = x_{e,k} \cdot w_{N+1}$, where $x_{e,k}$ is the share of packets blocked by the "deny all" rule that match $R_{e,k}$ and w_{N+1} is the "deny all" weight. We assume that the numbering of rules $R_{e,k}$ is arranged so that they are listed in order of decreasing values of $w_{e,k}$; that is, it is $w_{1,e} \geq w_{2,e} \geq \cdots \geq w_{M,e}$. Let $\mathbf{L}^*_e = [R_{e,1}, R_{e,2}, \ldots, R_{e,M}]$ be the new list.

4.2. Inserting Rule Extracted from Deny All String. This phase consists of the insertion into $\mathbf{R}^*$ of t rules $(0 \leq t \leq M)$, taken from $\mathbf{L}^*_e$. Thanks to the all disjoint relations among the rules in $\mathbf{L}^*_e$ and among these rules and the ones in $\mathbf{R}^*$, the extracted rules $R_{e,i}$ can be inserted in any position of $\mathbf{R}^*$ without generating conflicts.

Given the rule list $\mathbf{R}^*$, let its cost be

$$C(\mathbf{R}^*) = \sum_{k=1}^{N} k w_k + (N+1) w_{N+1}. \tag{6}$$

When $R_{e,i}$ is inserted into $\mathbf{R}^*$ with rank h the following cost is obtained:

$$\begin{aligned} C(\mathbf{R}^* \cup R_{e,i}) &= \sum_{k=1}^{h-1} k w_k + h w_{e,i} \\ &\quad + \sum_{k=h}^{N} (k+1) w_k + (N+2)(w_{N+1} - w_{e,i}) \\ &= C(\mathbf{R}^*) + \sum_{k=h}^{N+1} w_k - w_{e,i}(N+2-h). \end{aligned} \tag{7}$$

Equation (7) shows that $C(\mathbf{R}^* \cup R_{e,i})$ is a decreasing function of the weight $w_{e,i}$ for a given value of h. So, to reap the maximum gain (cost reduction), insertion should start from the extracted rule with the biggest weight. Once the optimum insertion location for this rule is found, the second

biggest weight extracted rule can be considered and so on. By virtue of the ordering of $\mathbf{L}_e^*$, the insertion algorithm starts by considering $R_{e,1}$ and finds a value for h_1, that is, the rank of $R_{e,1}$ in $\mathbf{R}^* \cup R_{e,1}$, which minimizes the overall rule list cost. To achieve this goal we should perform an exhaustive search. If the obtained minimum cost is less than the cost of the original list $\mathbf{R}^*$, then $\mathbf{R}^*$ is updated by adding the extracted rule $R_{e,1}$. The algorithm stores the updated list and its overall cost, and then it goes on evaluating the insertion of $R_{e,2}$ and so on, until it evaluates the insertion of all M rules of $\mathbf{L}_e^*$.

As a result of the insertion of extracted rules, we obtain M expanded rule lists, $\mathbf{R}_k^* \equiv \mathbf{R}^* \cup R_{e,1} \cup \cdots \cup R_{e,k}$ of length $N+k+1$ (including the "deny all" rule), where the k added rules have been assigned positions $h_1, \ldots, h_k$, respectively, $k = 1, \ldots, M$. The corresponding costs are denoted as $\Gamma_k \equiv C(\mathbf{R}_k^*)$; by extension, we set also $\Gamma_0 \equiv C(\mathbf{R}^*)$. Since any benefit brought by the insertion of $R_{e,k}$ grows up with $w_{e,k}$ and rules of $\mathbf{L}_e^*$ are ordered by decreasing weights, the sequence of obtained costs $\{\Gamma_k\}_{0 \le k \le M}$ is unimodal; that is, it has a unique minimum, say for index h^*. Then $h^* \ge 0$ is the optimum number of rules to be extracted from the "deny all" and it can be found at a cost linear with M.

5. Traffic Driven Adaptation of ACO

The traffic mix at the input of a packet filtering device changes over time, so that each rule is matched by a varying number of packets as new traffic flows set on or running ones end up. The changing traffic mix impacts ACO since the weights w_i of the rule list cost function defined in Section 4 are just the fraction of packets matching rule R_i.

To address this issue we follow the same approach developed, for example, in [5, 39]. We define an adaptive, event driven mechanism to trigger running of ACO, including the rule extraction from "deny all." The key elements of our proposed mechanism are (i) collection of device log information; (ii) statistical testing based on log data, to estimate traffic mix variation over time; (iii) extraction of rule from "deny all," provided the cost of the added rules is more than compensated by the processing gain.

The logic of ACO adaption is as follows. Let t_k be the last time that the rule list of the tagged device has been updated. Logs are collected from the packet filtering device, so that the management system can track the number $n_i(k)$ of packets matching rule R_i ($i = 1, \ldots, N$) and the overall number $n(k)$ of packets arrived at the device over the time interval of duration $[t_k, t_k + T_k]$. The collection time T_k is defined so as enough logs are accumulated to evaluate a statistically reliable estimate of the packet traffic fractions matching each rule; that is, $w_i(k) = n_i(k)/n(k)$, $i = 1, \ldots, N$, and $w_{N+1}(k) = 1 - \sum_{i=1}^{N} w_i(k)$. The weight vector $\mathbf{w}(k) = [w_1(k) \cdots w_{N+1}(k)]$ estimated at time $t_k + T_k$ is compared to the previous one, $\mathbf{w}(k-1)$, that has been used to optimize the rule list at time t_k. We test the hypothesis that the two weight vectors are drawn from the same probability distribution, by using the Chi Square test. If the hypothesis is inconsistent with the data (i.e., there is statistically reliable evidence that the input traffic mix has changed) a new optimization of the rule list is run, by taking the new weights equal to $\mathbf{w}(k)$. On the contrary, a new collection period starts and the whole process repeats all over again.

The automation algorithm is run individually for each device. The processing can be centralized in a network management system, by downloading logs accumulated by the filtering devices and storing them into the LogDB. The Algorithm 1 summarizes the steps carried out by the ACO Decision Support System (ACO-DSS) to adapt the rule list according to the filtered traffic mix. The ACO-DSS samples the LogDB, to check whether the number of packets $n(k)$ listed in the collected logs for the considered device in the kth sampling interval of duration T_k is larger than a threshold value $TH1$. If that is the case, the Chi Square statistical test is performed. If the test detects that the traffic mix has changed, ACO is run, including extraction of rules from "deny all." The performance gain of the resulting optimised list is assessed and compared with a threshold $TH3$. The new list is implemented if the performance gain is big enough.

The parameters $TH1$ and sampling time T_k can be dimensioned based on the following guidelines. Let us consider the kth sampling interval, drop the subscript k for simplicity, and let ϑ be the probability that a packet belongs to a given flow. The unbiased, asymptotically consistent estimator of ϑ is $\widehat{\vartheta} = x/n$, where x is the number of packets belonging to that flow out of the n logs collected in the considered interval. The relative root mean square error of this estimator is $\mathscr{E} = \mathrm{RMSE}(\widehat{\vartheta})/E[\widehat{\vartheta}] = \sqrt{(1-\vartheta)/(n\vartheta)} < 1/\sqrt{n\vartheta} \approx 1/\sqrt{x}$. This can be made less than a given error ϵ (we set 0.01), by taking x bigger than $1/\epsilon^2$ (10000 in our case). Accurate estimates of traffic flow rates are required especially for the largest flows, those that have the biggest impact on processing resources of the device so that their filtering can be optimized most profitably. Let ξ be the fraction of the device max throughput μ such that we want accurate estimates for those flows offering at least $\xi\mu$ pkts/s. Then, we should set T_k so that $\xi\mu T_k \ge 1/\epsilon^2$. For example, let $\mu = 42000$ pkts/s, as in Section 6, and let $\xi = 0.05$; that is, we aim at estimating accurately those flows whose rate is equal to or bigger than 5% of the device throughput. Then it must be $42000 \cdot 0.05 \cdot T_k = 10000$, whence $T_k \ge 4.65$ s. Even if the input rate of the input packet flow is two orders of magnitude less than the example above, still the requirement on T_k would be in the order of some hundred seconds. The fine tuning of T_k should be carried out in the specific networking environment where the packet filtering device is deployed. This issue is further discussed at the end of this section.

The decision about traffic mix changing exploits the *Chi Square test* (CST), to determine if the current sample weight vector belongs to the same probability distribution as the previous one (see also [39]). The choice of the significance level α, namely, the probability of false positive errors, is guided also by the observation that false positive errors are more critical than false negative errors. As a matter of fact, the latter implies that a real shift of traffic mix is overlooked: in that case all device rule lists stay the same so they might turn to be nonoptimized against the current traffic mix. In case of a false positive error, device rule lists would be updated

```
(1) for dev ← 1 to N_device do
(2)     if Timer(dev) expired then
(3)         n_j ←  # logs matching rule R_j for device dev
(4)         n ←  # logs collected for device dev
(5)         if n ≥ TH1 then
(6)             X_j ← n_j/n, j = 1, ..., Nrules(dev)
(7)             q ← Σ_{j=1}^{Nrules(dev)} (X_j − Y_j)^2/Y_j
(8)             if q > TH2 then
(9)                 Y_j ← X_j, j = 1, ..., Nrules(dev)
(10)                Extract rules from "deny all"
(11)                Optimize rule list of device dev
(12)                Evaluate percentage Cost Reduction pCR
(13)                if pCR ≥ TH3 then
(14)                    Upload optimized rule list into device dev
(15)                end if
(16)            end if
(17)        end if
(18)        Timer(dev) ← T
(19)    end if
(20) end for
```

Algorithm 1: ACO automation.

erroneously, since a traffic mix change is estimated whereas no actual change has occurred. The choice of the α level depends on error cost weighting of specific applications. We set $\alpha = 0.01$.

Let $X_j = n_j(k)$ be the number of logs matching rule R_j in the current observation interval and $Y_j = n_j(k-1)$ the number of logs for which the rule list is currently optimized. The test variable is

$$q = \sum_{j=1}^{N+1} \frac{\left(X_j - Y_j\right)^2}{Y_j}. \tag{8}$$

The null hypothesis is that the outcomes X_j are drawn from the same probability distribution as the Y_j, $j = 1, \ldots, N+1$. The test variable q in case of null hypothesis is asymptotically distributed as a Chi Square with N degrees of freedom for large sizes of the collected log sample. Hence q is compared with the Pearson threshold for the Chi Square test; namely, TH2 $\equiv \chi^2_{N,1-\alpha}$, the $(1-\alpha)$ quantile of the Chi Square random variable with N degrees of freedom. For $N \gg 1$, it is $\chi^2_{N,1-\alpha} \approx N + \Gamma_{1-\alpha}\sqrt{2N}$, where $\Gamma_{1-\alpha}$ is the $1-\alpha$ quantile of the standard Gaussian random variable. For $\alpha = 0.01$ it is $\Gamma_{99} = 2.576$.

If $q \leq$ TH2, the null hypothesis is accepted and the traffic mix is deemed to be unchanged. Then, the logs gathered in the last observation interval are discarded. In case the traffic mix is estimated to have changed, ACO is run, including the extraction of rules from "deny all." The amounts of obtained performance improvement do not necessarily justify the upload of the new rule lists. They are sent to the devices only if there is enough performance improvement to be gained. This is realized by means of threshold TH3, expressing the minimum percentage cost reduction (costRed%) that triggers upload of the new configuration into the DCDB and hence to the filtering devices (TH3 = 5%). The choice of TH3 is a trade-off between the benefits of the optimization and the costs of the configuration upload. These costs may be of different type, for example, unavailability of the device for a certain period of time (reset on upload), security issues, or reduced device redundancy. Note also that the benefits of the optimization may vary depending on the network devices and traffic, which is why TH3 should be chosen according to the specific scenario in which ACO is deployed.

A critical point for ACO automation to be feasible is the expected time scale of traffic mix variation. That depends on the specific networking context. We address specific examples in the next section, where the traffic mix changes due to a DoS attack that introduced abruptly new packet flows in an attempt to saturate the input capacity of the packet filtering device.

As an example of "ordinary" traffic variation over time (not affected by DoS attacks), we show in Figure 3 two traffic measurements taken from a tier-1 level Italian ISP operational public network (input and output traffic profiles are plotted on the positive and negative ordinates, resp.). The top graph reports the http/https traffic impacting a web portal of a major company. The traffic profile refers to a single IP address/port number (80) and is plotted in units of packets/s. The bottom graph shows UDP traffic impacting an authoritative DNS server (unique IP address/port number 53).

In both cases, it is apparent that significant changes of the volume of traffic of each flow occur over a time scale in the order of hours. This provides the opportunity to relax the requirement on the observation time interval to collect a reliable statistical sample of logs and determine when a

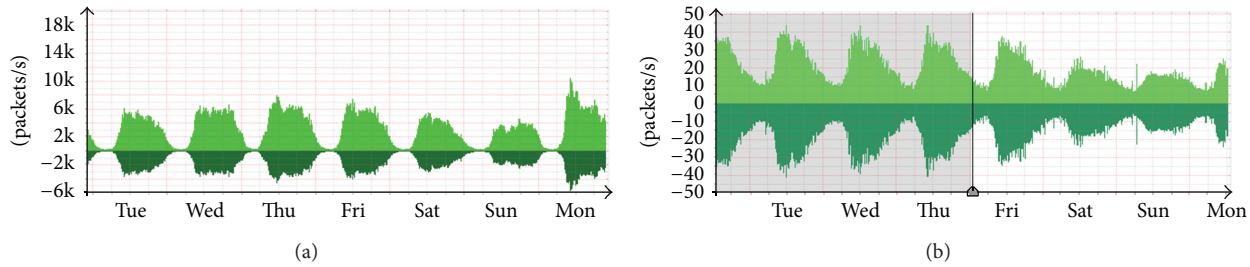


Figure 3: Examples of traffic flow time variation of a flow packet rate (pkts/sec) in public operational networks (packets in: positive y/packets out: negative y): (a) http/https flow in a major company web portal; (b) DNS traffic towards an authoritative DNS server.

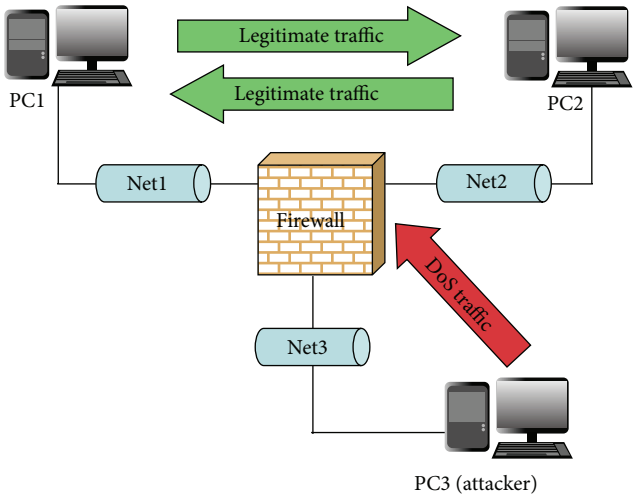


Figure 4: Test-bed layout for performance evaluation of ACO.

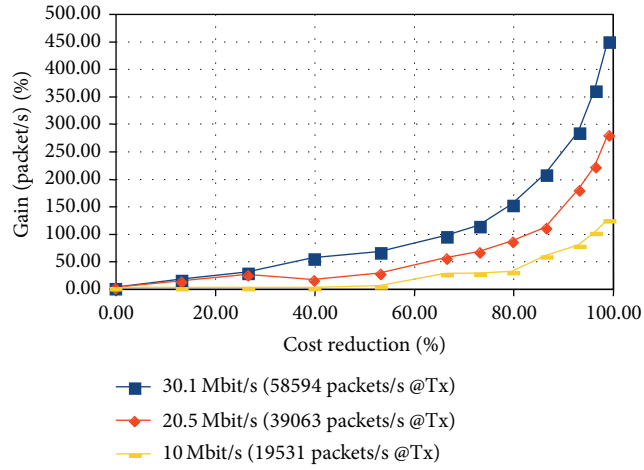


Figure 5: Filtering device throughput gain versus the tagged flow rule cost reduction for different values of the tagged flow inbound packet rate.

significant change occurs. It also relaxes the computational power requirements to run ACO.

6. Performance Evaluation of ACO

We carried out an experimental evaluation of the benefits of rule set optimization and rule extraction from "deny all." We set up a test-bed, outlined in Figure 4 and consisting of three Fast Ethernet subnets (physical link capacity: 100 Mbps). Two of them, *net1* and *net2*, are connected by a single packet filtering device Amtec SAS 1000, referred to simply as "filtering device" in the following. The device rules are configured so that only traffic between *net1* and *net2* is allowed. Attacking flows originate from *net3* and all of them match the "deny all" rule; hence they have the maximum possible processing cost.

The filtering device used in the experiments runs many security functions (i.e., known attacks detection, activity logging), which makes the test-bed a close picture of a real operational environment yet it forbids simple mathematical modeling of CPU activity. So, we run black box tests and we take packet loss ratio and packet throughput of a tagged flow through the filtering device as key performance indicators.

In Section 6.1 we discuss tests aimed at evaluating benefits of rule set optimization on processing performance of a packet filtering device. Section 6.2 deals with performance improvement by means of rule extraction from "deny all," specifically benefits in rejecting Denial of Service attacks.

6.1. Effect of Rule Cost Optimization. To evaluate the packet filtering device performance improvement obtainable as a function of the position of rules inside the list, we have generated UDP flow from *net1* to *net2* with a carried rate (throughput) λ_0 when the rank of the rule matching that flow is r_0. In Figure 5 we plot the throughput gain $(\lambda - \lambda_0)/\lambda_0$ as a function of the processing cost reduction $(r_0 - r)/r_0$, as the matching rule rank is decreased from r_0 to 1. Three different values of the inbound packet rates are considered. In all three considered cases, IP packets are 64 bytes long; it is $r_0 = 150$ and λ_0 ranges between 3200 and 3428 packets/s (Some dispersion of numerical results of experiments is due to the well known burstiness of traffic generation by means of IPERF [40].).

The results in Figure 5 show that the percentage throughput improvement grows with packet rate. This is a useful feature of ACO, since the demand for lowering the processing cost arises, when the traffic intensity increases. On the contrary, the less the inbound packet rate is, the less the optimization benefit is.

6.2. DoS Rejection Capability via Extraction of Rules from "Deny All". ACO can provide help in relieving the effect of DoS and DDoS attacks on the packet filtering devices. Denial of Service (possibly Distributed DoS) aims at overloading the CPU of the device by throwing a large amount of traffic on it, consisting of flows not envisaged in the policy design. These flows are discarded by virtue of the "deny all" rule, but this requires the entire list to be checked before a decision is taken on each packet. Even cache based accelerators can be ineffective, if a large number of different, undesired flows are thrown against a filtering device. That is not difficult to obtain, for example, by randomly changing source port, destination port, protocol type, or source address fields. ACO rule extraction from "deny all" can provide aggregated rules able to match the undesired traffic. Those rules can be merged in the rule list by the optimization procedure, so accounting for their weight in terms of matched packets.

ACO cannot be the only defence against DoS/DDoS attacks, especially when inbound link is saturated by anomalous traffic. In this case only the provider can definitely remove the effect of DoS/DDoS by disconnecting malicious sources of traffic. Despite that, we show that ACO is effective in detecting and reacting to DoS/DDoS attacks by relieving CPU load and protecting legitimate traffic.

Because of the limited number of associations that can be created and their single flow nature, cache based acceleration of processing works best with static traffic patterns. If a big surge of traffic made up of a large number of different and varying flows hits the filtering device, cache association is essentially ineffective. Extraction of rules from "deny all" as carried out in ACO aims at addressing this problem so as to complement the cache acceleration mechanism, by minimizing the time needed to match a packet. This is obtained by extracting maximally aggregated deny rules from the "deny all" and bringing them as close to the top of the rule set as dictated by the fraction of the inbound traffic hitting that rule.

The effectiveness of ACO is measured from a user point of view, as suggested in [41, 42], by injecting into the security device an allowed flow and measuring its degradation under the DoS attack. The considered types of legitimate traffic in our test-bed are TCP and UDP flows, as in [43], and FTP transactions. To measure network performances we take the following key performance indicators:

(i) long-term average net throughput for TCP and UDP;

(ii) average file transfer speed (in Mbit/s) for FTP.

For each type of legitimate traffic we vary the DoS attacking flow bit rate from 1 Mbit/s up to 35 Mbit/s. According to a worst case scenario, we set the attacking flow packet size to 40 bytes, so that attacking flow packet rates range from 3124 packets/s for a bit rate of 1 Mbit/s up to 110655 packets/s for a bit rate of 35 Mbit/s.

Results are shown in Figures 6, 7, and 8 for TCP, UDP, and FTP traffic, respectively. Each experiment consists of launching a legitimate flow. Let $t_0 = 0$ denote the start time of the experiment. All legitimate flows are set so that the filtering device processes them without any packet loss in case of no DoS attack. Performance worsening is only due to the onset of the attacking flow starting from time $t_1 = 200$ s. At time $t_2 = 400$ s ACO is run (The numerical values of these times are chosen to ease graph display; the reaction time of the automated ACO algorithm is in the orders of seconds; see Section 5.): a rule that captures the DoS flow is extracted from the "deny all" and the overall rule list is optimized as described in Section 4. The experiment run is stopped at time $t_3 = 600$ s.

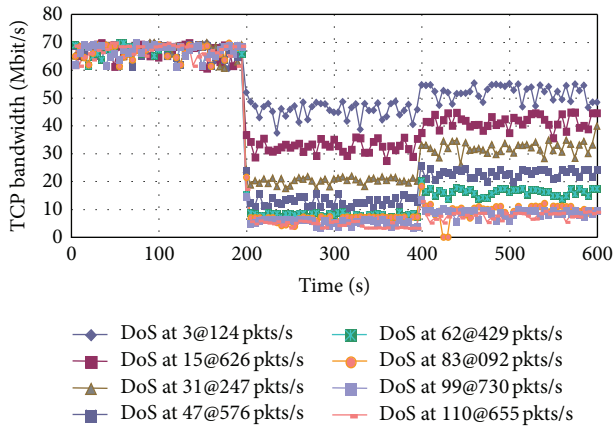


FIGURE 6: TCP sending rate sample path over 600 s, with DoS attack starting at time 200 s and ACO rule extraction and optimization carried out at time 400 s.

When the attack starts, performance of the legitimate flow degrades abruptly. After the extraction performed by ACO, it improves, in some cases getting back to the value observed prior to the attack. The legitimate flows react in different ways, according to the functionality of each protocol. For example, Figure 6 shows that TCP suffers major throughput loss even under a relatively mild attack (3124 pkts/s), due to TCP congestion window shrinking on packet loss detection. After ACO extraction of a rule filtering the attacking flow and optimization of the rule list, the device can process packets faster, thus reducing loss events and allowing TCP to attain a higher sending rate. UDP case is completely different (Figure 7), since there is no closed loop congestion control mechanism and datagram retransmission. In this case ACO extraction turns out to bring about a major performance improvement. The extraction phase of ACO is quite effective against DoS attack also in FTP case, as shown in Figure 8.

For each legitimate traffic and for each attack packet rate, we calculate the percentage improvement (PI) of the relevant performance indicator due to rule extraction from the "deny all" rule. PI of a given performance indicator X is defined as follows:

$$\mathrm{PI}_X = 100 \times \frac{E[X_A] - E[X_B]}{E[X_B]}, \tag{9}$$

where $E[X_B] \equiv$ Value of $E[X]$ *before* ACO execution and $E[X_A] \equiv$ Value of $E[X]$ *after* ACO execution.

The two average values are taken over 200 s time intervals. $E[X_A]$ is the average of the performance indicator from $t = 200$ s up to $t = 400$ s, whereas $E[X_B]$ is computed by

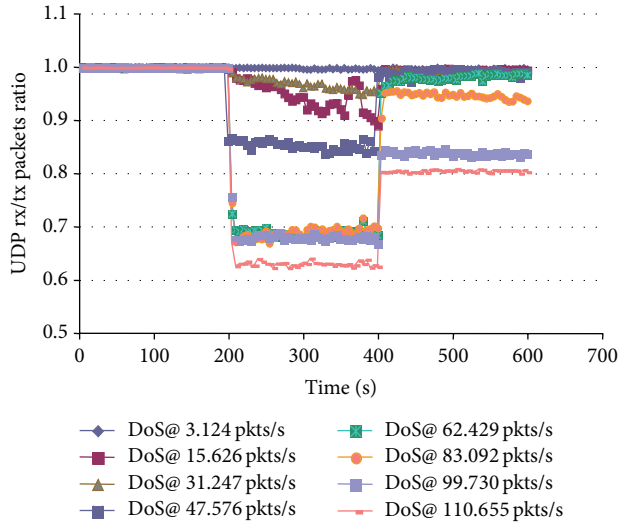


Figure 7: Sample path over 600 s of the ratio between sending and receiving rates of a UDP flow, with DoS attack starting at time 200 s and ACO rule extraction and optimization carried out at time 400 s.

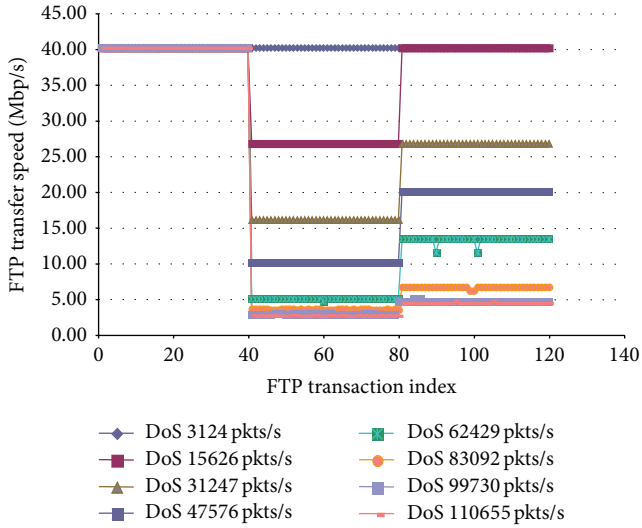


Figure 8: FTP transfer speed sample path for 120 transactions with DoS attack starting after 40 transactions and the extraction performed after 80 transactions.

averaging the performance indicator from $t = 400$ s up to $t = 600$ s. In the setup of these experiments, we force the execution of ACO at time $t = 400$ s, to let the time for stable regime be reached both before and after ACO execution.

In Figure 9 PI of the average download speed is plotted for FTP as a function of the attacking flow packet rate μ. Other cases are qualitatively similar to this one. For DoS attack at packet rates lower than about 6500 pkts/s the obtained PI is very low, so in those cases ACO rule extraction is not really needed. For bigger values of the attack flow packet rate the PI grows reaching a maximum for $\mu \approx 62400$ pkts/s and then it decreases somewhat, still hovering around 60%. Even under a heavy attack, performing ACO rule extraction and optimization allows users to download a file via FTP more than twice faster as compared to a nonoptimized rule list.

ACO can be also exploited against DDoS attacks, since the rules extracted from the "deny all" include aggregates of flows: they are actually the most general rules that cover the selector parameter subspace complementary with the subspace of allowed flows. So, a small set of rules can deal with all possible DDoS flows. When DDoS attack flows, possibly generated from different sources, match with a single extracted rule, the distributed attack is faced by ACO just as if it were a DoS attack from a single source. To demonstrate this robustness of our approach, we perform an experiment keeping the test methodology and network scenario same as before, except that three different attacking flows are generated in *net3*, originating from three different PCs. Attacking flows are such that a single extracted rule matches all attacking flows. For space reasons we do not show all DDoS test results, but just the PI for every legitimate flows (Table 1). Attacking flows aggregate bit rates used in the experiments are as high as about 80 Mbit/s. Even against such powerful attacks, the provision of ACO rule extraction and optimization reaps a

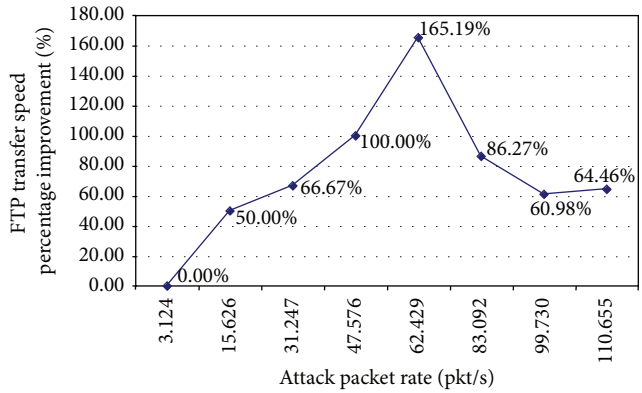


Figure 9: FTP transfer speed percentage improvement (PI) for average download speed after ACO rule extraction from "deny all" and optimization as a function of the attacking flow packet rate.

performance gain up to about 60% (FTP case) with respect to the degradation due to the attack.

7. Conclusions

This work focuses on optimization techniques for packet filtering devices such as firewall and security gateways. The basis of our proposal is the reduction of the packet processing cost relying on traffic observed on the network. Our tool collects traffic information by means of logs, sent by the managed devices, and exploits them to reorder the device rule set. Furthermore, it creates new rules extracted from the "deny all" rule to match input traffic flows that are not captured by other rules. This last feature can be useful against DoS/DDoS attacks.

Table 1: Percentage improvement (PI) of the efficiency parameter due to ACO rule extraction and optimization as a function of attacking flow packet rate, for TCP, UDP, and FTP legitimate flows.

μ [pkts/s]	TCP PI [%]	UDP PI [%]	FTP PI [%]
3124	—	0.04	0.00
6553	0.00	0.03	50.00
9830	27.29	0.47	50.00
13107	17.84	1.96	40.49
15626	20.93	3.61	66.67
31247	12.02	5.50	64.37
47576	26.58	17.55	72.79
62429	33.73	34.71	104.02
83092	21.87	32.92	64.51
99730	22.16	22.00	65.81
110655	20.61	23.65	64.71
131072	21.47	26.88	63.34
196608	18.83	26.59	68.42
262144	18.92	27.20	64.93

We have implemented ACO in an experimental testbed and measured the effect of ACO. Results point out that rules reordering entails a tangible improvement of packet filtering device processing performance. We have also tested the anti-DoS functionality of ACO extraction phase, measuring the attacks impact on legitimate traffic, and we have demonstrated that, for attacks with packet rate higher than a critical value, extracting rules from "deny all" allows legitimate users under attack to reach a performance improvement between 30% and 60% in most cases.

Appendices

A. ACO Procedure Details

ACO target can be stated as follows: find a ranking $\mathbf{y}$ that minimizes the rule list cost $C_{\mathbf{R}}(\mathbf{y})$ in (2), under the constraint that $y_i < y_j$ if $R_i \succ R_j$ in the DPT defined in Section 4. With reference to the DPT, we write $R_i \succ R_j$ if R_i belongs to the subtree rooted in R_j.

Let $\mathbf{R} = [R_1, \ldots, R_N, R_{N+1}]$ be the conflict-free rule list given as ACO input. We place the "deny all" rule on top of the DPT associated with $\mathbf{R}$ as a common root, thus formally making this structure a tree, which we refer to as Rule Tree (RT).

We define a reduction process of RT, starting from the leaf rules and reducing subtrees into single nodes, with an associated ordered *partial* rule list. The idea is that we start out with "atomic" lists made up of the single rule associated with each node of the DPT; then we visit the DPT starting from the leaves up to the root and we merge partial rule lists associated with visited nodes into a bigger list, with minimum cost and verifying the constraint. This procedure breaks up the problem of finding the optimal ordering of all rules into solving subproblems, consisting of merging two optimized lists into a bigger, still optimized one, until we end up with a list including all rules.

Let R_X denote the rule associated with node X of RT. Let $\mu_n(\mathbf{L}_1, \ldots, \mathbf{L}_n)$ be the *list merging function*, whose inputs are n (optimized) rule lists and output is a unique optimized ordered list encompassing all rules appearing in the n input lists. The steps of the reduction algorithm are as follows.

(1) *Initialization*. Set $k = 0$, $\mathcal{G}(0) \equiv$ RT, and $\mathbf{L}_X = [R_X]$ for each $X \in \mathcal{G}(0)$.

(2) *Leaf Node Reduction*. Take all leaf nodes $X_1, X_2, \ldots, X_S$, which are children of the same parent node Y in the tree $\mathcal{G}(k)$, remove them, and label Y with the ordered rule list $\mathbf{L}_Y = [\mu_S(\mathbf{L}_{X_1}, \mathbf{L}_{X_2}, \ldots, \mathbf{L}_{X_S}), R_Y]$; repeat until all (original) leaves of $\mathcal{G}(k)$ are removed.

(3) *Stop Condition*. Let $\mathcal{T}$ be the residual tree after the reduction in step (2) above. If the depth of $\mathcal{T}$ is greater than 1, let $\mathcal{G}(k+1) = \mathcal{T}$, replace k with $k+1$, and go back to step (2).

Since the depth of the RT is reduced by one at each step, the algorithm terminates in a finite number of steps, say K. The ordered list associated with the unique node of $\mathcal{G}(K)$ is the optimized rule list $\mathbf{R}^*$. The list merging function $\mu_n(\cdot)$ is applied to n *disjoint* rule lists $\mathbf{L}_j$ of length ℓ_j $(j = 1, \ldots, n)$ and yields a list $\mathbf{L}$ of length $\sum_j \ell_j$. For $n \geq 2$ it is defined recursively by

$$\mu_n(\mathbf{L}_1, \mathbf{L}_2, \ldots, \mathbf{L}_n) = \mu_2(\mu_{n-1}(\mathbf{L}_1, \mathbf{L}_2, \ldots, \mathbf{L}_{n-1}), \mathbf{L}_n), \quad \text{(A.1)}$$

with $\mu_1(\mathbf{L}_1) \equiv \mathbf{L}_1$. It suffices to specify the merge function for two lists, that is, $\mu_2(\mathbf{L}_a, \mathbf{L}_b)$. The function $\mu_2(\mathbf{L}_a, \mathbf{L}_b)$ merges the minimum cost lists $\mathbf{L}_a$ and $\mathbf{L}_b$ into a single minimum cost list $\mathbf{L}$ satisfying the DPT constraint, provided that both $\mathbf{L}_a$ and $\mathbf{L}_b$ separately do satisfy the same constraint.

The algorithm to form $\mathbf{L} = \mu_2(\mathbf{L}_a, \mathbf{L}_b)$ can be described inductively. Let $\mathbf{L}_a = [a_1, \ldots, a_n]$ and $\mathbf{L}_b = [b_1, \ldots, b_m]$ be two minimum cost lists, satisfying the DPT constraints. Let also $W_{a,k}$ denote the sum of weights of rules $[a_k, \ldots, a_n]$ for $k = 1, \ldots, n$, and let $W_{b,h}$ denote the sum of weights of rules $[b_h, \ldots, b_m]$, for $h = 1, \ldots, m$. Finally, let $\mathbf{M}$ be the matrix defined implicitly by the recurrence $M_{i,j} = \min\{W_{a,i} + M_{i,j+1}, W_{b,j} + M_{i+1,j}\}$ and $M(i, n+1) = M(m+1, j) = 0$ for $i = 1, \ldots, n+1$; $j = 1, \ldots, m+1$. Then consider the following.

(1) Set $t = 0$, $h = k = 1$, and $\mathbf{L}(0) = [\,]$ (empty list).

(2) If $W_{a,k} + M_{k,h+1} < W_{b,h} + M_{k+1,h}$, let $\mathbf{L}(t+1) = [\mathbf{L}(t), a_k]$ and replace k with $k+1$; else let $\mathbf{L}(t+1) = [\mathbf{L}(t), b_h]$ and replace h with $h+1$. Let $t = k+h-2$. If $(k \leq n) \wedge (h \leq m)$, repeat step (2).

(3) If $k = n+1$, let $\mathbf{L} = [\mathbf{L}(t), b_h, \ldots, b_m]$, and then stop.

(4) If $h = m+1$, let $\mathbf{L} = [\mathbf{L}(t), a_k, \ldots, a_n]$, and then stop.

In the following it is proved that this algorithm provides the optimum (minimum cost) list $\mathbf{L}$ if the two merged lists $\mathbf{L}_a$ and $\mathbf{L}_b$ are each separately optimized. Since $\mathbf{L}_a = [a_1, \ldots, a_n]$ and $\mathbf{L}_b = [b_1, \ldots, b_m]$ are each optimized, their ordering minimizes their respective costs:

$$C(\mathbf{L}_a) = \sum_{i=1}^{n} i w_{a,i}, \qquad C(\mathbf{L}_b) = \sum_{j=1}^{m} j w_{b,j}. \quad \text{(A.2)}$$

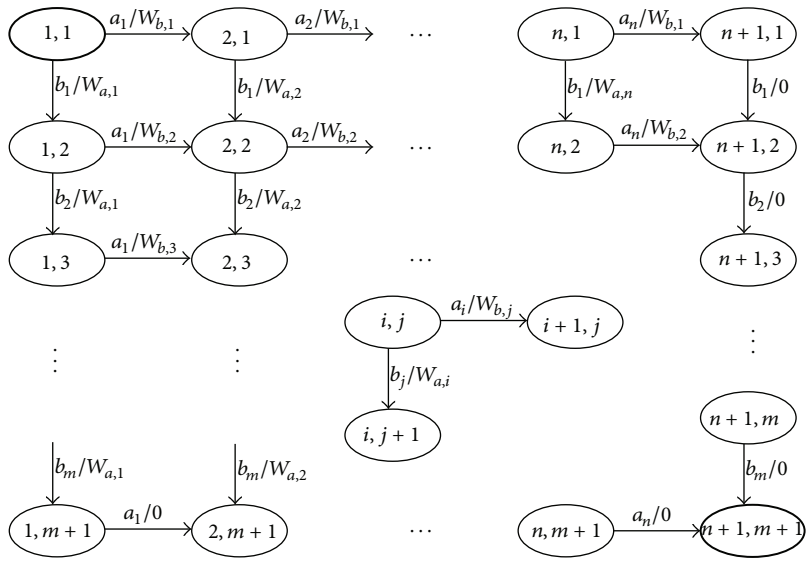


FIGURE 10: Graph for the optimization of the merging cost.

Let us now consider the merged list $\mathbf{L}$. Let $u_i \geq 0$ be the number of rules of $\mathbf{L}_b$ that are placed in between a_{i-1} and a_i, $i = 2, \ldots, n$; let $u_1 \geq 0$ be the number of rules of $\mathbf{L}_b$ that are placed before a_1; let $u_{n+1} \geq 0$ be the number of rules of $\mathbf{L}_b$ that are placed after a_n. Similarly, let $v_j \geq 0$ be the number of rules of $\mathbf{L}_a$ that are placed in between b_{j-1} and b_j, $j = 2, \ldots, m$; let $v_1 \geq 0$ be the number of rules of $\mathbf{L}_a$ that are placed before b_1, and let $v_{m+1} \geq 0$ be the number of rules of $\mathbf{L}_a$ that are placed after b_m. Note that $u_1 + \cdots + u_{n+1} = m$ and $v_1 + \cdots + v_{m+1} = n$. Then the cost of $\mathbf{L}$ can be expressed as

$$\begin{aligned} C(\mathbf{L}) &= \sum_{i=1}^{n} \left(i + \sum_{k=1}^{i} u_k \right) w_{a,i} + \sum_{j=1}^{m} \left(j + \sum_{k=1}^{j} v_k \right) w_{b,i} \\ &= C(\mathbf{L}_a) + C(\mathbf{L}_b) + \sum_{k=1}^{n} u_k \sum_{i=k}^{n} w_{a,i} + \sum_{k=1}^{m} v_k \sum_{j=k}^{m} w_{b,j} \\ &= C(\mathbf{L}_a) + C(\mathbf{L}_b) + \sum_{k=1}^{n} u_k W_{a,k} + \sum_{k=1}^{m} v_k W_{b,k}. \end{aligned} \tag{A.3}$$

Given the ordered, conflict-free, and optimized lists $\mathbf{L}_a$ and $\mathbf{L}_b$, the overall cost in (A.3) points out that minimization only depends on the two last sums (incremental cost due to merging). This merging cost can be rewritten in a different way. Let α_t be 1 if the tth element of $\mathbf{L}$ belongs to $\mathbf{L}_a$ and 0 otherwise, for $t = 1, \ldots, m+n$; let also $\beta_t = 1 - \alpha_t$. Let $\overline{C}_t = \alpha_t W_{b,B(t)} + \beta_t W_{a,A(t)}$, where $A(t) = \sum_{j=t+1}^{m+n} \alpha_j$ and $B(t) = \sum_{j=t+1}^{m+n} \beta_j$, for $t = 1, \ldots, m+n$. Then, (A.3) can be modified as follows:

$$\begin{aligned} C(\mathbf{L}) - C(\mathbf{L}_a) - C(\mathbf{L}_b) &= \sum_{k=1}^{n} u_k W_{a,k} + \sum_{k=1}^{m} v_k W_{b,k} \\ &= \sum_{t=1}^{m+n} \alpha_t W_{b,B(t)} + \beta_t W_{a,A(t)}, \end{aligned} \tag{A.4}$$

where the α_t's and β_t's result from the ordering of the merged list $\mathbf{L}$.

The merging of the two lists is done by preserving the relative order of rules belonging to the same original list, because of the conflict-free constraint. Then, the minimization of the merging cost can be done by selecting at step t which element to pick for the tth position of $\mathbf{L}$ from the current top elements of $\mathbf{L}_a(t)$ and $\mathbf{L}_b(t)$ in such a way that the sum in the rightmost side of (A.4) is minimized; $\mathbf{L}_a(t)$ and $\mathbf{L}_b(t)$ denote the lists obtained from $\mathbf{L}_a$ and $\mathbf{L}_b$ by deleting the elements already inserted in the first $t-1$ positions of $\mathbf{L}$ ($t = 1, \ldots, m+n$). The selection process is initialized with $\mathbf{L}_a(1) = \mathbf{L}_a$ and $\mathbf{L}_b(1) = \mathbf{L}_b$.

This problem can be restated as finding the minimum cost route from origin to destination nodes in the graph of Figure 10.

The state (i, j) of the graph refers to the (partial) lists $\mathbf{L}_a(i+j-2)$ and $\mathbf{L}_b(i+j-2)$, the top elements of the two lists being a_i and b_j, respectively. From state (i, j) a transition can be triggered to $(i+1, j)$ or $(i, j+1)$, in case a_i or b_j is selected, respectively. It is intended that state components $n+1$ or $m+1$ represent the end of the list. The labels on the graph arcs are coded as x/y where x denotes the popped up element and y is the arc cost.

The origin node is $(1, 1)$ and the destination node is $(n+1, m+1)$. The minimum cost route can be found, for example, by using a Bellman-Ford like approach, starting from the destination node. In general, let the minimum merging cost starting from state (i, j) be $M_{i,j}$. Then, from the graph structure it is easy to check that

$$\begin{aligned} M_{i,j} &= \min \left\{ W_{a,i} + M_{i,j+1}, W_{b,j} + M_{i+1,j} \right\}, \\ &\qquad i = 1, \ldots, n; \quad j = 1, \ldots, m; \\ M_{i,m+1} &= M_{i+1,m+1}, \quad i = 1, \ldots, n; \\ M_{n+1,j} &= M_{n+1,j+1}, \quad j = 1, \ldots, m; \\ M_{n+1,m+1} &= 0. \end{aligned} \tag{A.5}$$

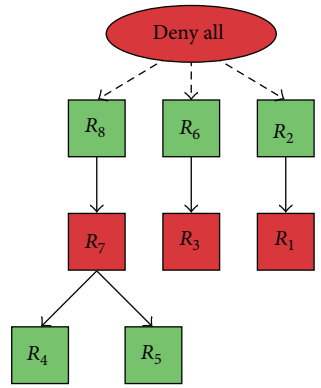


FIGURE 11: Device Pseudo-Tree associated with the rule list in Table 2. Red/green boxes denote *deny*/*allow* actions.

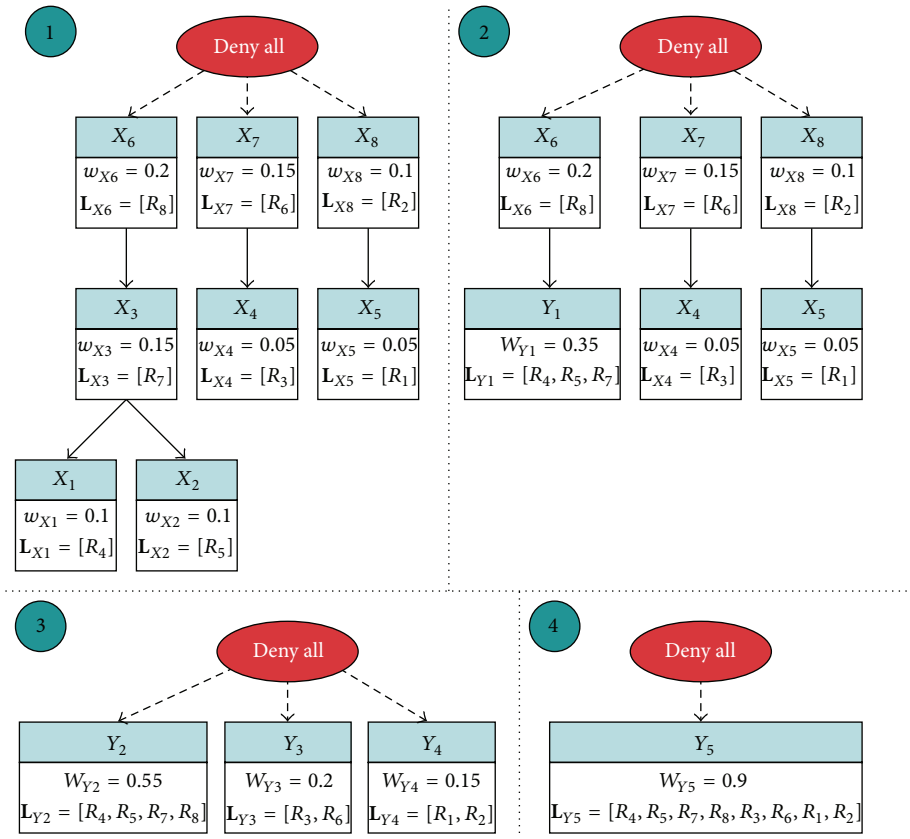


FIGURE 12: Optimization of the DPT of Figure 11 in four steps.

Starting from state (i, j), if $W_{a,i} + M_{i,j+1} < W_{b,j} + M_{i+1,j}$, then b_j is selected; otherwise a_i is selected. The complexity of the algorithm is linear with m and n.

B. Example of ACO Application

We develop a full blown example of application of ACO.

An example of conflict-free rule list that can be fed as input to ACO is given in Table 2.

Reordering must respect rule dependencies to avoid introducing conflicts. For example, if rule R_8 in Table 2 is brought to the top of the list because of its large cost, that creates a conflict with rule R_7, since the condition of R_7 is included in the condition of R_8 and their actions are opposite.

The DPT for the rule list in Table 2 is depicted in Figure 11. The "deny all" rule has been put on top of the DPT, as it is the most general rule.

The DPT of Figure 11 is used to optimize the rule list of Table 2 with the weights shown in the last column of Table 2. Figure 12 shows the optimization process in four steps (from left to right, from top to bottom). The final ordered, conflict-free, and optimized list is $\mathbf{R}^* =$

Table 2: Example of rule list with $N = 8$ (R_9 is the "deny all" rule).

R_i	Source IP address	Destination IP address	Destination port	Source port	Protocol type	Action	w_i
R_1	3.0.1.120/32	0.0.0.0	0-65535	0-65535	Any	Deny	0.05
R_2	3.0.1.0/24	0.0.0.0	0-65535	0-65535	TCP	Allow	0.1
R_3	2.0.0.1/32	0.0.0.0	80	80	TCP	Deny	0.05
R_4	3.0.0.2/32	0.0.0.0	0-1024	0-1024	UDP	Allow	0.1
R_5	3.0.0.3/32	0.0.0.0	0-1024	0-1024	TCP	Allow	0.1
R_6	2.0.0.0/24	0.0.0.0	0-65535	0-65535	Any	Allow	0.15
R_7	3.0.0.0/24	0.0.0.0	0-65535	1024-65535	Any	Deny	0.15
R_8	3.0.0.0/24	0.0.0.0	0-65535	0-65535	Any	Allow	0.2
R_9	0.0.0.0	0.0.0.0	0-65535	0-65535	Any	Deny	0.1

Table 3: List of rules extractable from the "deny all" rule of the rule set in Table 2.

$R_{e,k}$	Destination IP address	Source IP address	Destination port	Source port	Protocol type	Action	$x_{e,k}$
$R_{e,1}$	3.0.2.0-255.255.255.255	0.0.0.0	0-65535	0-65535	Any	Deny	0.7
$R_{e,2}$	2.0.1.0-2.255.255.255	0.0.0.0	0-65535	0-65535	Any	Deny	0
$R_{e,3}$	0.0.0.1-1.255.255.255	0.0.0.0	0-65535	0-65535	Any	Deny	0.3

Table 4: Variation of cost when extracted rule $R_{e,1}$, with weight $w_{e,1} = 0.7\rho$, is inserted into the optimized rule list $\mathbf{R}^*$ with rank h; the weight of the "deny all" rule is parametrized by $\rho \in [0.1]$.

R_k	$\widetilde{w}_k$	rank h of $R_{e,1}$	$\Delta C(h) = \sum_{k=h}^{N+1} (\widetilde{w}_k - \widetilde{w}_{e,1})$
R_4	$0.1111 \cdot (1 - \rho)$	1	$1.0000 - 6.3000\rho$
R_5	$0.1111 \cdot (1 - \rho)$	2	$0.8889 - 5.4889\rho$
R_7	$0.1667 \cdot (1 - \rho)$	3	$0.7778 - 4.6778\rho$
R_8	$0.2222 \cdot (1 - \rho)$	4	$0.6111 - 3.8111\rho$
R_3	$0.0556 \cdot (1 - \rho)$	5	$0.3889 - 2.8889\rho$
R_6	$0.1667 \cdot (1 - \rho)$	6	$0.3333 - 2.1333\rho$
R_1	$0.0556 \cdot (1 - \rho)$	7	$0.1667 - 1.2667\rho$
R_2	$0.1111 \cdot (1 - \rho)$	8	$0.1111 - 0.5111\rho$
R_9	ρ	9	$0.3\,\rho$

$[R_2, R_3, R_5, R_6, R_1, R_4, R_7, R_8, R_9]$. Its overall cost is $C(\mathbf{R}^*) = 4.75$, to be compared with the initial cost $C(\mathbf{R}) = 5.75$ (17.4% cost reduction).

As an example of how a list of rules extracted from "deny all" can be created, we refer to the rule list in Table 2. The correspondent DPT is shown in Figure 11. Table 3 illustrates the set of rules $R_{e,k}$ extractable from the "deny all" rule of the list in Table 2 and the associated normalized weights $x_{e,k}$. So, the list $\mathbf{L}_e^*$ of candidate rules for extraction is $\mathbf{L}_e^* = [R_{e,1}, R_{e,3}]$, since $R_{e,2}$ has 0 weight.

If we apply "deny all" rule extraction to the rule list of Table 2, by using the extracted rule set of Table 3, it turns out that there is no cost reduction. This is because rule extraction has a useful impact only if the number of packets matching "deny all" is a significant fraction of the overall packets dealt with by the filtering device. In the example of Table 2 "deny all" traffic accounts for just 10%. As another example, let us assume that the weight of "deny all" is ρ and other weights stay the same except they are scaled to make the sum of all weights equal to 1. The new weights are denoted with a tilde and are shown in Table 4. The last column of Table 4 reports the difference between the cost of the rule list $\mathbf{R}^*$ and the one with rule $R_{e,1}$ inserted with rank h; namely, $\Delta C(h) \equiv C(\mathbf{R}^* \cup R_{e,1}$ with rank $h) - C(\mathbf{R}^*)$. The weight of the extracted rule is $w_{e,1} = 0.7\rho$, according to the first line of Table 3. It is easily found that the most convenient rank for $R_{e,1}$ is $h = 1$ for all $\rho > 0.1792$, which is the intersection point between the straight lines corresponding to the first and fifth rules. For example, for $\rho = 0.25$, the cost $C(\mathbf{R}^* \cup R_{e,1}$ with rank $1) = C(\mathbf{R}^*) - 0.575$ where the cost of the list with no extracted rule is $C(\mathbf{R}^*) = 5.46$.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] S. Acharya, J. Wang, Z. Ge, T. Znati, and A. Greenberg, "Simulation study of firewalls to aid improved performance," in *Proceedings of the 39th Annual Simulation Symposium*, pp. 18–25, Huntsville, Ala, USA, April 2006.

[2] S. Acharya, J. Wang, Z. Ge, T. F. Znati, and A. Greenberg, "Traffic-aware firewall optimization strategies," in *Proceedings of the IEEE International Conference on Communications (ICC '06)*, pp. 2225–2230, Istanbul, Turkey, July 2006.

[3] S. Acharya, M. Abliz, B. Mills, and T. Znati, "Optwall: a hierarchical traffic-aware firewall," in *Proceedings of 14th Annual Network and Distributed System Security Symposium (NDSS '07)*, San Diego, Calif, USA, February 2007.

[4] L. Zhao, Y. Inoue, and H. Yamamoto, "Delay reduction for linear-search based packet filters," in *Proceedings of the International Technical Conference on Circuits/Systems, Computers and Communication (ITC-CSCC '04)*, Matsushima, Japan, July 2004.

[5] H. Named and E. Al-Shaer, "Dynamic rule-ordering optimization for high-speed firewall filtering," in *Proceedings of the ACM Symposium on Information, Computer and Communications*

Security (ASIACCS '06), pp. 332–342, Taipei, Taiwan, March 2006.

[6] K. Golnabi, R. K. Min, L. Khan, and E. Al-Shaer, "Analysis of firewall policy rules using data mining techniques," in *Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS '06)*, pp. 305–315, Vancouver, Canada, April 2006.

[7] A. Hari, S. Suri, and G. Parulkar, "Detecting and resolving packet filter conflicts," in *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM '00)*, pp. 1203–1212, Tel Aviv, Israel, March 2000.

[8] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 10, pp. 2069–2083, 2005.

[9] E. Al-Shaer and H. Hamed, "Firewall policy advisor for anomaly detection and rule editing," in *Proceedings of IEEE/IFIP Integrated Management Conference (IM '03)*, Colorado Springs, Colo, USA, March 2003.

[10] S. Ferraresi, S. Pesic, L. Trazza, and A. Baiocchi, "Automatic conflict analysis and resolution of traffic filtering policy for firewall and security gateway," in *Proceedings of the IEEE International Conference on Communications (ICC '07)*, pp. 1304–1310, Glasgow, Scotland, June 2007.

[11] S. Ferraresi, E. Francocci, A. Quaglini, and F. Picasso, "Security policies tuning among IP devices," in *Knowledge-Based Intelligent Information and Engineering Systems*, vol. 4693 of *Lecture Notes in Computer Science*, pp. 149–158, Springer, Berlin, Germany, 2007.

[12] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen, "Policy core information model version 1 specification," Tech. Rep. RFC-3060, 2013.

[13] A. Westerinen, J. Schnizlein, J. Strassner et al., "Terminology for policy-based management," Tech. Rep. RFC-3198, 2001.

[14] C. Basile and A. Lioy, "Towards an algebraic approach to solve policy conflicts," in *Proceedings of the Workshop on Logical Foundations of an Adaptive Security Infrastructure (WOLFASI '04)*, Turku, Finland, July 2004.

[15] Q. Duan and E. Al-Shaer, "Traffic-aware dynamic firewall policy management: techniques and applications," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 73–79, 2013.

[16] A. Tapdiya and E. W. Fulp, "Towards optimal firewall rule ordering utilizing directed acyclical graphs," in *Proceedings of the 18th International Conference on Computer Communications and Networks (ICCCN '09)*, pp. 1–6, San Francisco, Calif, USA, August 2009.

[17] Y.-K. Chang, C.-C. Su, Y.-C. Lin, and S.-Y. Hsieh, "Efficient gray-code-based range encoding schemes for packet classification in TCAM," *IEEE/ACM Transactions on Networking*, vol. 21, no. 4, pp. 1201–1214, 2013.

[18] H. Lim, N. Lee, G. Jin, J. Lee, Y. Choi, and C. Yim, "Boundary cutting for packet classification," *IEEE/ACM Transactions on Networking*, vol. 22, no. 2, pp. 443–456, 2014.

[19] Z. Wu, M. Xie, and H. Wang, "Design and implementation of a fast dynamic packet filter," *IEEE/ACM Transactions on Networking*, vol. 19, no. 5, pp. 1405–1419, 2011.

[20] H. Lim, Y. Choe, M. Shim, and J. Lee, "A quad-trie conditionally merged with a decision tree for packet classification," *IEEE Communications Letters*, vol. 18, no. 4, pp. 676–679, 2014.

[21] L. Abeni, N. Bonelli, and G. Procissi, "Randomized packet filtering through specialized partitioning of rulesets," *IEEE Communications Letters*, vol. 17, no. 12, pp. 2380–2383, 2013.

[22] G. Misherghi, L. Yuan, Z. Su, C.-N. Chuah, and H. Chen, "A general framework for benchmarking firewall optimization techniques," *IEEE Transactions on Network and Service Management*, vol. 5, no. 4, pp. 227–238, 2008.

[23] H. Hamed, A. El-Atawy, and E. Al-Shaer, "On dynamic optimization of packet matching in high-speed firewalls," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 10, pp. 1817–1830, 2006.

[24] A. El-Atawy, T. Samak, E. Al-Shaer, and L. Hong, "Using online traffic statistical matching for optimizing packet filtering performance," in *Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM '07)*, pp. 866–874, Anchorage, Alaska, USA, May 2007.

[25] A. El-Atawy and E. Al-Shaer, "Adaptive early packet filtering for defending firewalls against DoS attacks," in *Proceedings of the 28th IEEE Conference on Computer Communications (INFOCOM '09)*, pp. 2437–2445, Rio de Janeiro, Brazil, April 2009.

[26] A. Hussain, J. Heidemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '03)*, Karlsruhe, Germany, August 2003.

[27] "Information processing systems, open systems interconnection basic reference model, part 2: security architecture," Tech. Rep. ISO 7498-2, 1989.

[28] D. Moore, G. M. Voelker, and S. Savage, "Inferring Internet Denial-of-Service Activity," University of California at San Diego—CAIDA, 2001, http://www.caida.org/publications/papers/2001/BackScatter/.

[29] C.-H. Shen and T.-Y. Chung, "PFC: a new high-performance packet filter cache," in *Proceedings of the International Computer Symposium*, Taipei, Taiwan, December 2004.

[30] W. Jiang and V. K. Prasanna, "Large-scale wire-speed packet classification on FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '09)*, pp. 219–228, Monterey, Calif, USA, February 2009.

[31] Y. Qi, J. Fong, W. Jiang, B. Xu, J. Li, and V. Prasanna, "Multi-dimensional packet classification on FPGA: 100 Gbps and beyond," in *Proceedings of the International Conference on Field-Programmable Technology (FPT '10)*, pp. 241–248, Beijing, China, December 2010.

[32] V. Pus, J. Blaho, and J. Korenek, "Memory optimizations for packet classification algorithms in FPGA," in *Proceedings of the 13th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS '10)*, pp. 297–300, April 2010.

[33] W.-G. Wang, T. Zhang, Y.-F. Zheng, and Y. Yang, "Realization of FPGA-based packet classification in embedded system," in *Proceedings of the IEEE Intrumentation and Measurement Technology Conference (I2MTC '09)*, pp. 943–942, Singapore, May 2009.

[34] A. Begel, S. McCanne, and S. L. Graham, "BPF+: exploiting global data-flow optimization in a generalized packet filter architecture," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '99)*, pp. 123–134, Cambridge, Mass, USA, August-September 1999.

[35] P. Rolando, R. Sisto, and F. Risso, "SPAF: stateless FSA-based packet filters," *IEEE/ACM Transactions on Networking*, vol. 19, no. 1, pp. 14–27, 2011.

[36] C. Lonvick, "The BSD syslog protocol," Tech. Rep. RFC-3164, 2001.

[37] D. New and M. Rose, "Reliable delivery for syslog," Tech. Rep. RFC-3195, 2001.

[38] A. Rizzi, M. Panella, and F. M. F. Mascioli, "Adaptive resolution min-max classifiers," *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 402–414, 2002.

[39] W. Wang, H. Chen, J. Chen, and B. Liu, "Firewall rule ordering based on statistical model," in *Proceedings of the International Conference on Computer Engineering and Technology (ICCET '09)*, pp. 185–188, Singapore.

[40] A. Botta, A. Dainotti, and A. Pescapé, "Do you trust your software-based traffic generator?" *IEEE Communications Magazine*, vol. 48, no. 9, pp. 158–165, 2010.

[41] J. Mirkovic, A. Hussain, B. Wilson et al., "Towards user centric metrics for denial-of-service measurement," in *Proceedings of the Workshop on Experimental Computer Science*, June 2007.

[42] J. Mirkovic, A. Hussain, S. Fahmy, P. Reiher, and R. K. Thomas, "Accurately measuring denial of service in simulation and testbed experiments," *IEEE Transactions on Dependable and Secure Computing*, vol. 6, no. 2, pp. 81–95, 2009.

[43] J. Mirkovic, P. Reiher, C. Papadopoulos et al., "Testing a collaborative DDoS defense in a red team/blue team exercise," *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 57, no. 8, pp. 1098–1112, 2008.