Adaptive Early Packet Filtering for Defending Firewalls against DoS Attacks

Adel El-Atawy , Ehab Al-Shaer
School of Computing
DePaul University
Chicago, Illinois, USA
Email: {aelatawy,ehab}@cs.depaul.edu

Tung Tran, Raouf Boutaba
School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
Email: {t3tran@,rboutaba@bbcr.}uwaterloo.ca

***Abstract*—A major threat to data networks is based on the fact that some traffic can be expensive to classify and filter as it will undergo a longer than average list of filtering rules before being rejected by the default deny rule. An attacker with some information about the access-control list (ACL) deployed at a firewall or an intrusion detection and prevention system (IDS/IPS) can craft packets that will have maximum cost. Most optimizations made to current filtering techniques target the accepted traffic.**

In this paper, we present a techniques that is light weight, traffic-adaptive and can be deployed on top of any filtering mechanism to pre-filter unwanted expensive traffic. The technique utilizes Internet traffic characteristics coupled with special carefully tuned representation of the policy to generate early defense policies. We use Boolean expressions built as BDDs to represent relaxed versions of the policy that are faster to evaluate. Moreover, it is guaranteed that the technique will not add an overhead that will not be compensated by the gain in filtering time in the underlying filtering method. Evaluation has shown considerable savings to the overall filtering process, thus saving the firewall processing power and increase overall throughput. Also, the overhead changes according to the traffic behavior, and can be tuned to guarantee its worst case time cost.

I. Introduction

The overall performance and reliability of network-based operations depend on the reliability and stability of its perimeter network security devices. Firewalls and intrusion detection/prevention systems (IDS/IPS) are the cornerstone of a network's security infrastructure. If an attacker becomes capable of attacking these devices then the whole network can be dramatically affected. By overwhelming firewalls with packets that are chosen to be very hard to process, the attacker can reduce the overall throughput drastically and the payoff to the attacker's bandwidth is multiplied by the ratio of the worst to the average packet matching cost. Recent advancements in probing/scanning techniques showed that it is possible for an intelligent scanner to find out a great deal about what the firewall policy looks like by a reasonable number of scanning packets [12], [24].

Routers and firewalls classify packets according to prefix tables and access control lists where each packet is matched against the policy to decide the appropriate action. The same task takes place in Intrusion Detection Systems (IDS) where the header and, often, the payload of the packet are matched against a set of patterns and conditions that trigger alerts regarding malicious activities. A filtering technique that is capable of performing regardless of the number of criteria fields will prove useful in a wide variety of devices.

In this paper, we will describe an early filtering/decision technique that reduces the packet matching cost by a dynamically changed pre-filtering phase that resides before the original firewall matching technique. Special consideration was given to having minimal on-line operations and reasonable overhead periodic maintenance. While in this paper we focus on firewalls, the technique is applicable to any device that performs packet matching based on a predefined deterministic policy. The actual implementation was written and deployed as a plug-in for the standard IPTables firewall. The technique is triggered prior to the start of the regular matching, and if a packet needs further processing it is passed to the next layer, otherwise it is directly dropped or accepted to the system.

Firewall rules are often written as exceptions to the default deny rule for incoming traffic. This explains the research emphasis on optimizing the acceptance decision path in firewall filtering. However, some rejected packets might traverse long decision paths of rule matching before they are finally rejected by the default-deny rule. This causes significant matching overhead that in most cases increases with the number of rules in the firewall policy. Many of such packets can be rejected with minimal cost if the appropriate conditions were checked first. For example, checking the most-significant bit of the destination IP can eliminate all incoming traffic if it contains the wrong IP class (*e.g.*, all class A domains has a 0 in the MSB).

The proposed technique uses a modified representation of the policy to attain its goal. The policy is compiled into a single Boolean expression that represents its acceptance space. In this representation, each bit in the packet header is considered an input binary variable into the Boolean expression, and only packets that evaluate this expression to true are passed through the firewall successfully. This expression, however, is quite complex and evaluating it for every packet can be a considerable overhead. Thus, we aim to simplify this expression by placing an upper bound on the depth by which we can traverse the expression tree to evaluate the packet. In other words, our policy is an approximation that is aware of its limitations.

In the following sections, we present some of the related work (Section II), then the technique will be explained in detail and analyzed in Section III. After that, in Section VI, the technique's performance is evaluated and discussion about its applicability is shown. Finally, we end with final remarks, conclusion and future work in Section VII.

II. Related Work and Background

A. Optimizing Packet Classification

The problem of packet classification and filtering has been studied extensively, and various promising results has been around for years. Recently, the idea of having a technique that is traffic aware has started to get some attention in the literature (some of these techniques are referenced below [6], [9], [15], [16], [20]). It is important to note that most of the previous work target the acceptance path, where legitimate packets are to be accepted as fast as possible.

Policy Simplification: An important direction is minimizing the policy (access-control list) itself, by finding another smaller set of rules with the same semantics. In [2], the problem was mapped to minimizing the number of opaque rectangular patches that are needed to come up with a picture with a desired pattern. The problem is NP-hard and the authors provided a poly-time approximation algorithm ($\rho = O(min(n^{1/3}, OPT^{1/2}))$). Same problem was addressed in [27] where adjacent rules satisfying a set of simple criteria can be converted into a fewer number of rules. However, these two approaches add extra dependencies between rules, hindering the optimization of the filtering operation.

General Optimization Techniques: Many approaches have been studied for the general packet classification problem. Mainly, one or more of the following approaches are used: hardware-based optimization [4], [22], [23], specialized data structures and geometric algorithms [10], [25], and several heuristics [6], [13], [14], [26].

Hardware-based solutions using Content Addressable Memories(CAM) exploit the parallelism in the hardware but face scalability problems due to cost, power and size limitations of CAMs. Also, the nature of their deployment and filter-to-HW mapping makes such techniques limited to specific filter types (routing vs. IDS filters). In [22], rules are structured as a trie, with classification time linear in size of filter bits. In [4], Aggregated Bit Vector (ABV) are used to reduce the problem to dimension lookup instead of bit lookup thus enhancing the complexity. They use d independent lookups on each dimension, followed by a combining phase. In [23], Prefix Inclusion Coding (PIC) is used to compress the representation of the policy when mapped to TCAM entries. This solves to some extent the scalability issue in number of rules, but not in the number of filtering fields. Other researchers addressed the question of whether TCAMs are the only solution for fast classification or not [3], [7].

Srinivasan et al. [25] used a table of field value cross-products and pre-compute the earliest rule matching each cross-product. Obviously, the size of these tables grow dramatically with the number of rules. Geometric structures were also used with promising results, as these technique proposed by Feldmann et al. [10]. They used Fat Inverted Segment (FIS) Trees, partitioning dimensions recursively based on rule endpoints. This method scales well with the number of filtering rules, but not with the number of dimensions (*i.e.*, a problem for extended firewalls or IDSs).

Decision-tree based classification algorithms based on geometric cutting was used by Gupta and McKeown [13] and Woo [26] where both schemes build a decision tree optimized based on greedy choices. Woo [26]'s approach uses multiple decision trees, thus increasing search time while reducing storage while Gupta and McKeown's Recursive Flow Classification (RFC) helps pipeline the matching on the expense of scalability. Similarly, the Hierarchical Cuttings (HiCuts) scheme described in [14] uses range checks instead of bit tests at each node of the decision tree. In [6], decision trees were also used with common-branch adjustment to reduce space requirements. However, the data elements in the decision trees are whole rules, and the choices used to build the tree are greedy choices.

Although all previous work contribute significantly to the advancement of packet classification research, their main objective was to improve the worst-case matching performance with less emphasis on average case gains. Hence, they are not traffic aware and in many cases they exhibit high space complexity.

Traffic-aware and driven Techniques: Utilizing traffic characteristics to the optimization process was addressed by many researchers in the field (*e.g.*, the work by Gupta [15], Fulp [11], Hamed [16] and El-Atawy [9]). By introducing statistical data structures in optimizing packet filtering, these papers became among the most interesting foundation publications in this domain. In the first paper, depth-constrained alphabetic trees are used to reduce lookup time of destination IP addresses of packets against entries in the routing table. As the focus of this paper is routing lookups, the scheme is limited on search trees of a single field with arbitrary statistics. In [16] the work was extended to multiple fields in firewall policies with the capability of parallel processing. The authors in [9] proposed a technique that is based on a specialized policy encoding (*i.e.*, policy segments) in order to build Huffman trees that adapt to the traffic statistics. The technique can also be parallelized and its worst case could be bounded. An approach to find an optimal ordering of rules while maintaining policy semantics was addressed in [17]. The maintained form of the rules makes it a plausible preprocessing phase to any other technique. In [7], a hybrid approach between software and hardware was proposed, it also incorporates the traffic statistics to dynamically build new rules in the form of a cache. These new rules have better hit ratio than using original rules from the rule set.

B. Early Rejection via FV-SC

Another tightly related traffic aware technique is the Field Value Set Cover (FV-SC) technique we presented in [16]. We introduced the basic idea of early rejection with a preliminary

design of the technique using field value set covering. The technique targets the traffic that will eventually hit the default rule. The technique used custom statistics-induced rules that should match such traffic with minimal overhead to other flows. The analysis was limited to filtering engines that use linear matching for the original policy rules, while in this paper we elaborate using different matching cost patterns. Moreover, in the current paper, we exploit our experience in representing rules and policies (*i.e.*, access-control entries and lists) as Boolean expressions. Another limitations of the previous work is that it focused on rejection paths only, while in this paper the technique can find shortcuts for both accepted and denied traffic.

The intuition behind it was that some field values used in the policy are more popular than others. Therefore, it is possible to find a small set of them that every accepting rule has to contain at least one element (*i.e.*, hence the name "set cover"). Obviously, if a packet does not match any of the these values, then it can be safely dropped. For example, if all accept rules use as destination a certain subnet or allow a certain destination port number, then packets that do not have neither can be safely rejected without any further matching. The nature of the problem makes finding the optimal set cover not required and more than one solution will be needed to cover the policy with varying set-covers. Using one set-cover solution, a Rejection Rule (RR) is formed, such that each element will me translated to a Rejection term (RT) that together compose RR.

$$RR = \bigwedge_{S_k^j \in A} (Pkt(f_j) \neq v_k) \tag{1}$$

where $Pkt(f_j)$ is the value of field f_j in the packet to be inspected. A typical rule can look like: $RR = (DPort \neq 80) \wedge (DPort \neq 20) \wedge (DAddr \neq 15.16.17.18) \wedge (Proto \neq UDP)$

III. Relaxed Policy Matching

Our technique "Early Filtering" (EF) is based on approximating the policy with another while having an error that can be deterministically rectified. Provided a packet, the technique evaluates it against the policy, and reaches one of three options: Either the packet should be accepted, rejected or more filtering is needed by the original policy. The original policy is still being deployed using the filtering method implemented in the firewall, but it is not executed unless the early filtering module fails to reach a conclusion.

A. Policy Representation and Modeling

To attain our goal of representing the policy and efficiently approximating it, we will use the Boolean expression representation of the policy, as used in [8], [9]. For example, assume having a simple packet header that consists of only source and destination fields each is just 3-bits long. Table I shows two examples of such rules converted to Boolean expression conversion. After converting each rule, the expression representing the accept space of the whole policy (Φ) will be compiled as follows;

TABLE I
Example of rule-to-Boolean expression conversion

Rule	src	dst	expression
R_1	11*	0**	$x_0x_1\overline{x_3}$
R_2	1**	01*	$x_0\overline{x_3}x_4$

$$\Phi = \bigvee_{i=1}^{n} [\phi(i) \bigwedge_{j=1}^{i-1} \neg\phi(j)]$$

where $\phi(i)$ is the Boolean expression version of rule i, and n is the total number of rules in the policy. This policy expression incorporates the first-rule priority matching rule (*i.e.*, rule i matches a packet if the packet does not match any higher rule). Evaluating this function by simple substitution of variables by their values from the packet header will result in the correct classification result (*e.g.*, "allow" or "drop" in the case of firewall policies).

The implementation and maintenance of this expression can get quite complex. Therefore, we the Binary Decision Diagrams data structure (BDD) for its representation. BDDs can facilitate the matching by representing the expression in the form of a tree, where each variable is needed to be checked only once. Thus, the overall matching cost is bounded by the number of bits needed to represent the fields used to build the matching criteria. In the case of standard firewalls, this sums up to 104 variables (32*2 for IP addresses, 16*2 for the ports and 8 for the protocol). Any packet filtering technique, in order to be of some value, has to beat the cost of 104 bit-comparisons per packet (*i.e.*, compare-and-branch cost at each node). Our technique is based on using only shallow leaves to approximate the policy, while leaving longer decision paths for the second stage where ordinary/legacy packet filtering techniques can take over.

According to the traffic statistics, we can obtain the range of depths to go into the policy expression tree (*i.e.*, BDD-represented expression) while maintaining a positive gain in performance. If the traffic that hits the shallow leaves (*i.e.*, close to the root) in the BDD tree is high enough, the gain will be valuable. As this percentage gets lower the technique will be less effective to the point that it just introduces overhead with no gain, and in such case the algorithm will automatically shutdown the early filtering path.

B. Preprocessing Phase

The technique uses an off-line step where the user-provided text policy is converted into a single compound Boolean expression. For example, assuming the simple 3 bit addressing scheme used above, if we are given the policy in Table II with all rules "allow"; the overall expression would be $\Phi = x_1x_2 + \overline{x_1}x_2x_4\overline{x_4} + \overline{x_1}x_2\overline{x_3}x_4 + \overline{x_1}x_2\overline{x_3}x_4x_5 + \overline{x_1x_2}x_3$.

Conceptually, we build BDD trees representing several approximation levels concurrently: $\Phi_0, \Phi_1, \ldots \Phi_n$. Each tree (or equivalently expression) Φ_i approximates the original policy using the first i variables. Without replicating the storage,

	src	dst
R_1	11*	***
R_2	011	0**
R_3	010	1**
R_4	001	***
R_5	010	00*

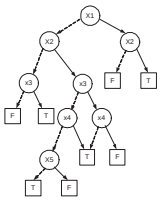


TABLE II
A SAMPLE POLICY OF A SIMPLISTIC SYSTEM AND ITS GENERATED BOOLEAN TREE

just limiting traversal depth while evaluation will attain the same goal with a single tree. Furthermore, each node will be associated with an integer that represents the first tree level at which a decision can be reached (*i.e.*, number of hops until the first leaf). For example, if the traversal is limited to the first 10 levels, the matching process will terminate if it reached a node with marked value of 18.

C. Packet Classification and Maintenance

Upon packet arrival, the fields used in classification are extracted from the packet header and sorted according to their order in the expression tree, so they can be used one-by-one in navigating the tree. This step can be implemented in hardware, as a permutation of incoming bits in parallel to their required order. Hard-wiring is even possible to attain very low latency. Tree navigation is itself a very simple set of instructions; check the variable at the current node, load a certain integer if true (*i.e.*, left child node entry in the BDD table) and another if false. This is repeated until a node is reached having a final value instead of a variable ID or reaching the maximum depth allowed in the tree.

In order to use the optimal tree depth for the current traffic statistics, random sampling takes place to check if the traffic will perform better using another depth limit that is deeper or shallower. The goal is to investigate this with minimal overhead as possible. In Algorithm 1, this checking is performed smoothly within normal filtering operations. Upon packet arrival, a depth is chosen randomly such that the current depth has the highest probability being selected, and the probability decays linearly in both directions: up and down. A constant ρ specifies the probability by which we leave the optimal depth limit to try other values.

1) Dynamic approximation level selection: In this section, we will show how we can make our system dynamic to traffic properties and based on the policy structure, and previous performance measures. Using both the policy information and traffic statistics we can determine the upper bound on the number of levels to be used in the Early Filtering phase. It is clear that the more levels, the more likelihood to decide traffic as each level tend to cover more policy space if it contained leaf nodes. So, the space covered with increasing levels is a non-decreasing function. As a start, let us assume for simplicity that all levels have the same probability of deciding a packet, and the underlying policy has a constant

Algorithm 1 Early Filtering operation

Calculating the depth (current Depth = D):

$$\text{trial\_D} = \begin{cases} D & prob. = 1 - 2 * \rho \\ D + i & prob. = \frac{2\rho(D_{max} - D - i)}{(D_{max} - D)^2} \\ D - i & prob. = \frac{2\rho(D - i)}{D^2} \end{cases}$$

$d = 0$
$node = root$
while $(d < trial\_D) \wedge (node \neq leaf)$ **do**
  if $node.detailDepth > trial\_D$ **then**
    break
  end if
  if $(B[node.var) = true)$ **then**
    $node \leftarrow node.left$
  else
    $node \leftarrow node.right$
  end if
end while
Update statistics:
INC T_j, $j = 1 \ldots d$ {Total packets tested with D levels}
if node = leaf **then** {Decision was reached at level d}
  INC $\triangle\delta_d$
end if

cost matching. Now, take δ_l as the portion of traffic that will be decided using l levels, and δ_{inf} as the maximum percentage of the traffic that can be early filtered. In our current implementation this is exactly 100% of the traffic as all filtering criteria are used in our expression tree. However, if some fields are not implemented, then this figure can be less than unity. Then for the early filtering levels to decrease the average number of comparisons, we use the same derivation from [16] to reach that the number of levels l should be governed by;

$$\begin{aligned} \frac{n}{2}(1 - \delta_{\text{inf}}) + n\delta_{\text{inf}} > & \frac{l}{2}\delta_l + (l + \frac{n}{2})(1 - \delta_{\text{inf}}) \\ & + (l + n)(\delta_{\text{inf}} - \delta_l) \end{aligned}$$

This leads to (when $\delta_{inf} = 1$):

$$l < \frac{2n\delta_l}{2 - \delta_l} \tag{2}$$

The inequality represents the necessary condition of having the average number of comparisons per packet without using early filtering, more than average cost of filtering by the early filtering tree, combined with the regular underlying matching technique. This implies that even for a perfect case where all traffic was decided by l levels, we can still have a limit on the number of levels to use. More complete analysis is provided in Section V.

IV. IMPLEMENTATION ISSUES

Here, we discuss some of the issues that will affect the performance/functionality of the technique. However, these points were postponed for the sake of clarity.

A. Implementation and deployment

The technique was implemented as a plug-in for IPTables that allows user-space processes to handle packets for pre-processing. We used the BuDDy package [1] for the im-

plementation of the BDD operations that handles the policy expression tree. When the user process starts, it retrieves the policy from IPTable, compile it into a single expression, and stores this for packet-per-packet processing. For each arriving packet our module will be consulted first before given to the full policy, where it gets the chance to early make the decision for the packet.

The packet matching takes place by traversing the policy expression from the root for each packet. At each node, the operation is very lightweight as it consists of a single bit comparison and branch. This can be implemented with a very short sequence of machine instructions. Given a node in the expression tree, two lookups are needed to retrieve byte and bit position that match a variable in the expression ($XLAT$ instruction), a bit test against packet header information (BT instruction), and a few assignment and conditional jump instruction (MOV and Jx). These instructions sum up in the core of the traversal to 45 clocks [18]. Therefore, for full matching, assuming maximum allowable depth of the tree, the early filtering module will be able to handle 45K packets per second using just 5% of a 3GHz Intel CPU (assuming 80 levels into the tree is the maximum needed). More details about the implementation can be found in the authors' project page.

Deployment in general will be least invasive to the original firewall (or other similar devices as IDSs, and routers), as they are implemented as an insertion into the processing flow without affecting any other modules. Early filtering processing takes place right before the normal filtering operations, and their statistics measures are saved separately. Therefore, the only items need to be changed from a user perspective is enabling the feature, and reading the gain due to using it via a single hit percentage parameter. From a developer's point of view, the technique will be implemented, and extra storage for the monitoring module will be allocated for keeping track of the technique applicability to current traffic patterns. However, this statistics is common to be collected by default in packet matching modules.

B. Changing the depth of the expression tree

To obtain the optimal depth of the relaxed expression depth tree to follow the changing traffic statistics, we have to compare the current depth, with adjacent levels. This can be performed by sampling packets that will undergo the complete processing cycle using the larger tree, and comparing the cost with the currently used tree. However, checking only one depth higher and lower might cause the technique to get stuck in local minima. Therefore, in the algorithm, we give the depth a probability distribution by which it can try different depths from 0 (disabling the early filtering) to D_{max} (using all the layers in the tree). The pdf used is a triangular distribution with the peak at the currently used depth D, and tapers out linearly to both extremes. In the next section, we will lay down technique and rule efficiency criteria that will be used for updating the search structure. To use formulas in the next section, each packet will contribute to the counters of one of the depths, and periodically Eqn. 5 can be used to decide whether we should use this depth or not. The tree depth that minimizes the expression for the average number of comparisons (as in Eqn. 3), is to be used in the next time window.

C. Node annotations for traversal skipping

If a tree is truncated at a certain level, all nodes at this level will have their subtrees removed and a decision cannot be made at this path. If two such nodes are siblings, then their parent will be also indecisive. This can propagate higher in the tree. For example, if a tree is a full tree, and all the leaves have alternative values. Then any approximation for this tree will be indecisive for all paths. An early hint will be beneficial at some nodes to identify such parts of the tree where actual traversing will not lead to a final decision for the packet.

In the preprocessing phase, we save with each node the minimum depth at which it will start to become beneficial to traverse through ($detailDepth$ in Algorithm 1). At actual matching time, if the current used depth is less than a certain node's detail depth, then the early filtering will terminate at this node with a request for normal classification. This will add an extra comparison per node for the sake of providing shortcuts that can offset the added cost. This decision is to be made by the vendor of the device depending on the device configuration, and expected policy size to use in such a device.

Variable reordering for optimal BDD processing: The order by which we check for variable values in the BDD can be of extreme importance to the performance of the whole technique. However, finding the optimal ordering can be a taunting task, formally it has been proven to be an NP-Complete problem [5]. It is a main point in our pending future tasks in this research work. However, from previous analysis of field value distribution and the way IP addresses and subnet masks are specified [16], [21], one can reach an educated guess on how a "good" variable ordering should look like. The BDD tree is smallest in size when effective variables are kept on top of the tree. In other words, decisions on bits that have a lower discriminating power will force an overlap to occur between the two subtrees. Here, we present some of the points considered when assigning the currently used variable order:

- *MSB vs LSB*: Specifying ranges of values for any field will lower the effect of the lower-significant bits in discriminating between packets. Therefore, variables mapped to MSBs should always precede LSBs for any field.
- *Source vs Destination*: Destination fields tend to be more specified in firewall rules (and essentially routers) than source fields (*e.g.*, in some configurations, source ports might never be specified).
- *Contiguous variables*: There is no need to make variables representing the same field be contiguous. Separating them might be better for the overall structure. For example, the destination address might be best represented if its bits (MSB down to LSB) be mapped to variables: 6-9,11-24,30,32,41-52.

Field	dIP	proto	sIP	dPort	dIP
Field bits	(0,15)	(0,7)	(0,7)	(0,7)	(16,23)
Variables	0,15	16,23	24,31	32,39	40,47
dPort	dIP	sPort	sIP	sPort	sIP
(8,15)	(24,31)	(0,7)	(8,15)	(8,15)	(16,31)
48,55	56,63	64,71	72,89	80,95	96,103

TABLE III
VARIABLE ALLOCATION TO FILTER FIELDS. *F(x,y)* DENOTES VARIABLES FROM X TO Y INCLUSIVE OF FIELD *F*, COUTING FROM MSB AS ZERO.

Splitting a data block into separate non-contiguous blocks of variables for BDD building has been used before with excellent results in different fields. For example, in [19] variable ordering was performed in an adhoc manner by interleaving variables from multiple operands for the purpose of sequential circuit verification.

V. ANALAYSIS AND EFFICIENCY CRITERIA

In this section, we will show how to quantify the efficiency of the technique proposed. The analysis considers adding more traversal levels in the Boolean expression tree equivalent to adding more rules in the first technique with a linear matching cost pattern. This is due to way we traverse the tree, where maximum depth is the worst case cost for traversing the tree of the EF. Therefore, we will be using the notion of a EF rule interchangeably with levels to denote a matching step in our techniques (*i.e.*, rejection rules (as in FV-SC) or expression tree layers).

In order to determine the effect/worthiness of adding a specific level at run time, more analysis is needed. After using D levels in the expression tree, we have α_D, δ_D, and γ_D be the traffic portion decided by the policy , decided for by the D EF levels, and the default rule respectively. Now, we can state the average number of comparisons/matching per packet after using D levels as follows:

$$A_D = c.D(\delta_D + \alpha_D + \gamma_D) + n(\frac{\alpha + \beta_D}{2} + \gamma_D) \quad (3)$$

where c is the relative evaluation cost of single BDD traversal to that of a normal filtering rule which is usually in favor of EF (*i.e.*, single bit comparison versus complete rule matching). As the matching algorithm 1 shows, traversing a tree based on bit comparison can be greatly optimized. In the Implementation, an example of a typical execution timing figures are provided. We also have $\partial\delta/\partial D > 0$, $\partial\alpha/\partial D, \partial\gamma/\partial D < 0$, and $\alpha_D + \gamma_D + \delta_D = 1$. Let $\triangle\delta_D$ be the portion of the total traffic that is decided by the D^{th} level. Then we can simply show that

$$\beta_{D-1} + \gamma_{D-1} = \alpha_D + \gamma_D + \triangle\delta_D \quad (4)$$

To justify including the D^{th} level: $A_D - A_{D-1} < 0$ must hold. Thus, using (3) and (4) we derive the following condition:

$$\frac{\triangle\delta_D}{c} > \frac{\frac{\alpha_D}{2} + \gamma_D}{n} = \frac{c\,(1 - \delta_D + \gamma_D)}{2n} \quad (5)$$

Algorithm 2 Dynamic Level Selection

for $i = 1 \ldots n$ **do**
  $\triangle\delta_i = \triangle\delta_i / T_j$ {Normalize}
end for
$D = min_i i * \triangle\delta_i + c.M_i/T_i$ {Depth with min avg cost}
if $D < overhead\_cost$ **then**
  Disable Technique
end if

Either form can be used to facilitate evaluation at run time according to the type of statistics kept at the firewall. It is worth mentioning that the last derivation assumed that the average number of levels and the original rules that a packet will go through before finding a match is simply $D/2$ or $n/2$, respectively. However, this is not generally the case; the average might be quite different from half of the rule count. In this case, the conditions placed on evaluating levels will differ by this new ratio as well (*i.e.*, the constant in the eqn 5 will be adjusted accordingly). Formally,

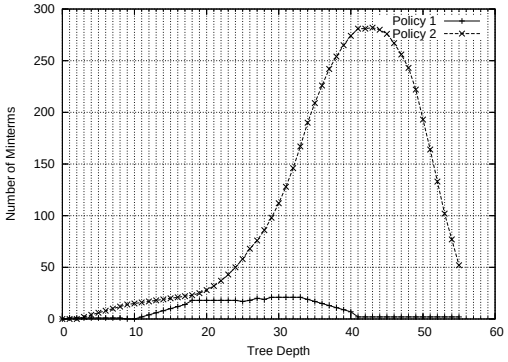
$$\triangle\delta_D > \frac{c\,(1 - \delta_D + \gamma_D)}{a_D n}$$

where a_D is a constant that depends on the actual number of average rules matched. It increases with increasing the average number of levels or number of rules matched on average in the EF or the regular matching respectively.

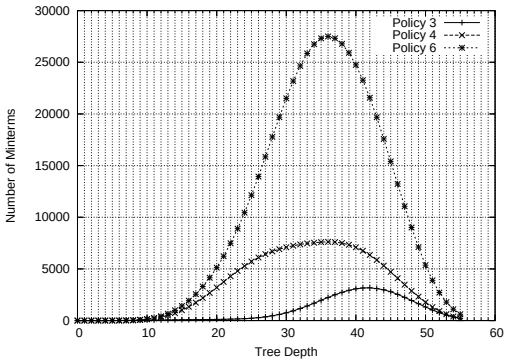
After each window of time, the added rule can be evaluated based on (5) to decide whether the D^{th} level is to be included or removed, as described in Algorithm 2.

Similar analysis is needed for other techniques with better performance than sequential matching. What affects our analysis is the time complexity of these techniques and the relative computational time of a single comparison in these techniques to ours (*i.e.*, the cost of a single RR evaluation). The latter was already incorporated into the factor c. The complexity formula of the underlying filtering technique will change the form of the condition formulae slightly.

- *Logarithmic Time* These category cover most of the technique shown in Section II. These include techniques based on balanced search trees like alphabet trees used in [15], [16] and Huffman trees in [9]. Also geometric-based search structures follow the logarithmic complexity class.
  Eqn. 2 becomes $D < \frac{2\delta_D \log(n)}{2 - delta_D}$ and Eqn. 5 is changed to $\Delta\delta_D > \frac{c(1-\delta_{D-1}/2)}{2\log n - cD}$ Which shows that as the number of levels already used increase, it is getting harder to add more. The same goes for relative computational cost.
- *Constant Time* Many of the hardware implemented/optimized techniques belong to this category. Also, techniques with search structures that does not depend on the policy size, like hash-based techniques. Some techniques that place decision only at the leaves of tries and trees that does not grow with policy size have constant running time, as in [22].
  For such algorithms Eqn 2 is $D < \frac{2\delta_D}{c(2-\delta_D)}$ and Eqn. 5 is



(a) Smaller policies



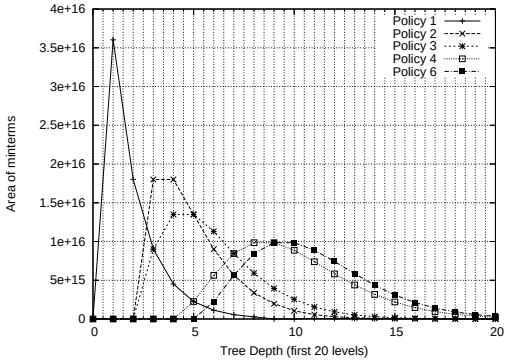
(b) larger policies

Fig. 1. The frequency distribution of minterms in the negated policy expression

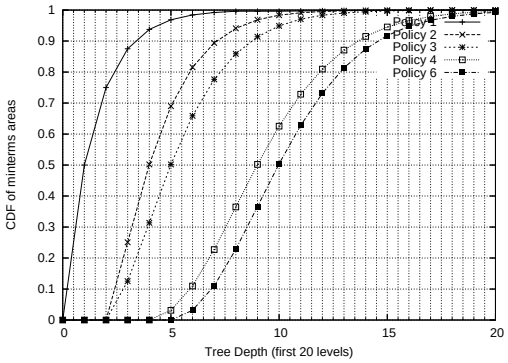
changed to $\Delta\delta_D > \frac{2c(1-\delta_{D-1}/2)}{2+cD}$. This shows that there is still a feasible range of $\Delta\delta_D$ in which the benefit of early filtering levels will be sensed even with constant cost filtering techniques. However, this largely depend on the exact implementation of the filtering technique, and the traffic behavior.

VI. Performance evaluation

One of the main concerns for the applicability of the technique is how common is having short min-terms (*i.e.*, short paths to a leaf in the expression tree). The first two charts (Fig. 1-a and b) show two groups of policies with their minterm frequency distribution (separated into two charts just for clarity). As we can see as the policy size increases, the number of minterms with larger size (*i.e.*, higher number of involved variables) increases. This is intuitive for the fact that larger policies are written mainly to handle more cases, and some of them are inevitably special cases and are very restricted. It is important to emphasize that if these experiments were performed on the negated version of the policy, the frequency distribution of minterms lengths would have been different. A minor optimization step can be performed at the startup phase, which is to select the version (*i.e.*, positive or negated) that has less complexity. It is worth noting that the BDD representation of the policy will be identical except at the terminal leaves that will be swapped. Therefore, although it will not affect the space requirements, it is a cheap operation and can be performed even for a minor gain.



(a) Minterms area distribution



(b) CDF of the Minterms area as a percentage of the total policy area

Fig. 2. The distribution of area of minterms in the negated policy expression

In Fig. 2, we can see how these different tree levels contribute to the overall deny space of the policy (*i.e.*, the space where rejected packets belong). Most of the space is covered by the first few levels of the expression tree. With the assumption of uniformity of rejected traffic, we can get a sense of the amount of savings possible using this approach. In the first policy, we can reject more than 90% of the traffic using just 4 bit comparisons. The second and third policies can be protected by early filtering tree of depth 5 that will remove at least 50% of the unwanted incoming traffic. For larger policies (*i.e.*, thousands of rules), we will need a higher number of levels but still the savings are guaranteed using the proposed technique due to the fact that it selects that best tree depth to use adaptively and following the traffic dynamics.

Figure 3 shows the average number of rules matched for packets that passed the early filtering module. A 1000 rule IPTables policy was deployed at the filtering host for Figures 3, 4 and 5. There is a slight tendency of increased cost, as these packets are already hard to filter for the given policy. However, Figure 4, shows the average cost in IPTables matches over all packets arrived at the machine. A significant reduction in matching cost is obvious when EF started to engage with depth higher than 5 levels. When levels is 30 or more, IPTables does not seem to receive any significant number of packet to match reducing the average cost to almost zero. On the other side the average cost per packet for packets in the EF module is shown in 5, where it reaches the maximum when it is 0 or

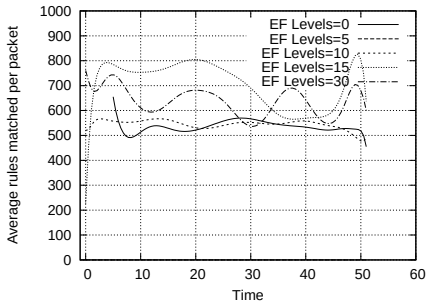


Fig. 3. Average number of matches per packet forwarded to IPTables.

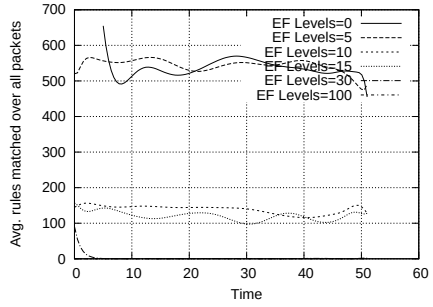


Fig. 4. Average number of matches over all packets received.

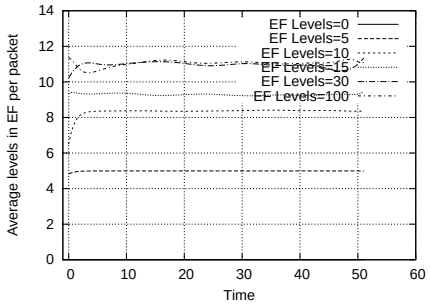


Fig. 5. Average number of levels needed for EF.

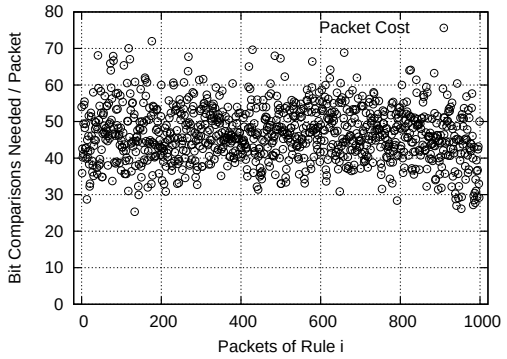


Fig. 6. Depth needed to reach a decision for packets targeted to each rule of a 1000 IPTables policy. Shows the independence between rule location and needed depth in tree to reach a decision for its packets.

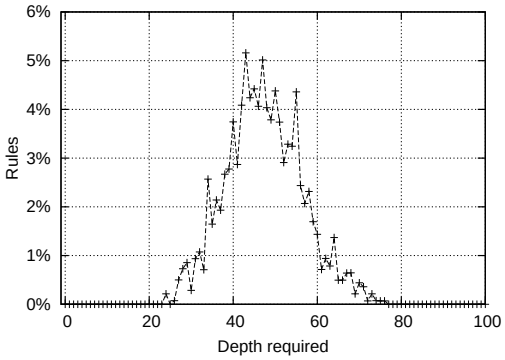


Fig. 7. Distribution of rules whose traffic requiring a certain depth to be completely filtered by EF. A 1500 rule policy is used.

5. For higher levels, some of the packets did not require the whole number of allowable levels thus reducing the average from 10,15, 30,100 to approximately 8, 9, 11 and 11 levels respectively.

Figure 6 shows a very important advantage of our technique: Orthogonality of rule order in original policy and cost of matching its traffic in our technique. For this experiment, we generated an artificial trace that will hit all rules in turn. By measuring the needed cost (without a bound on traversal), we obtained the shown graph. The obvious randomness shows that the benefit for traffic targeted to the last few rules of the original policy will benefit dramatically from adding our early filtering module. Moreover, the randomness in the cost is even more beneficial when denial of service attacks are considered. An attacker will not be able to identify the type of flows will be more expensive for the firewall to match. Even if the complete information about the policy is available to the adversary, she will not be able to identify those expensive flows unless all other details are also available; mainly the variable ordering used to build the BDD. This last aspect increases the difficulty for the attacker to figure the correct order put of the possible $n!$ where n is the number of variables in the definition (*i.e.*, 126 variables for representing TCP, UDP, ICMP with their options/flags). The same conclusion can be clearly seen in Figure 7. Using an 1500 rule policy, and counting the number of rules whose packets will be matched via a certain

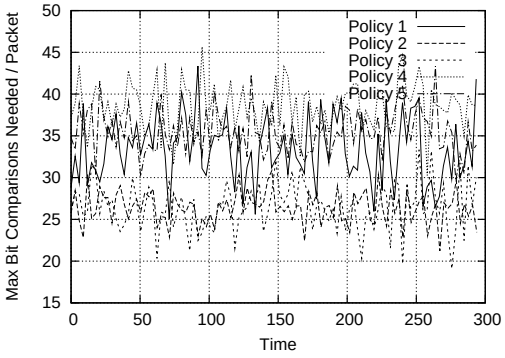


Fig. 8. Depth needed to reach a decision for packet traces injected to 5 different IPTables policies (sizes: 100,500,1000,1500,5000)

depth shows a typical normal distribution. This emphasizes the fact the our technique effectively hides the information of rule order from an attacker, and makes it impossible for the adversary (even with complete policy information) to obtain a better than random guess about what rules to target in order to achieve maximum damage to the firewall performance.

In Figure 8, we show the cost over time of matching multiple traces against firewall policies of different sizes. The policies shown span a size from 10 rules up to 5000 rules. As we can see there is no significant tendency towards increasing the average cost of matching over all packets in the trace by increasing the policy size. This removes the burden from

the filtering engine and shows that our technique is highly scalable. As the policy size increase it will get easier for the EF technique to prove useful.

VII. Conclusion

In this paper we address the possibility of Early Filtering traffic in firewalls and similar network security devices. We introduced a novel technique: Relaxed Policy Expression. The technique is shown to provide an efficient filter for unwanted packets or easy to accept/reject packets based on the policy definition and the statistics of the incoming traffic. The implementation shows promising performance and practically plausible space requirements, besides it is possible to be deployed on and be optimized for different platforms. It was evaluated with typical policies and in real implementation of IPTables. The technique gave impressive results in shown experiments, but the fact that it depends on the specific implementation of the Boolean expression module makes it open for even further improvements as this module can be optimized via hardware special support. Using the currently all-software implementation of the BDD package, the module was implemented and attached to the IPTables filtering sequence. Using policies with a range of sizes from tens of rules to 5000 rules, the system was evaluated and it has shown that a gain in performance. In almost all policies investigated, more than 40% of the denied traffic was early rejected via less than 30 bit comparisons. In summary, the technique is: 1) light weight, 2) its gain is guaranteed to be worth the overhead, 3) it hides the policy rule order from any attacker even if he is capable of monitoring packet-by-packet processing time, and 4) simple and easy to implement.

Further research is needed to place theoretical bounds on the gain of such techniques. Also, hardware acceleration can give a tremendous boost to the applicability of early filtering methods. Other early filtering algorithms can be developed with better dynamic properties as well as more capability in cooperating with existing packet filtering techniques. For example, providing hints to the following packet classification layer as what to expect and what kind of traffic has been filtered out or allowed to pass through the early filtering module. This can also allow the underlying filtering algorithm to check the packet against a smaller subset of the policy. In our specific implementation, this translates to marking packet not matched by our module so that they will matched against specific shorter chains rather than the whole policy. In summary, the topic is far from being exhausted with respect to developing new techniques and knowing the possible performance gains.

References

[1] Buddy: Bdd c++ package. "http://sourceforge.net/projects/buddy", 2006.

[2] David A. Applegate, Gruia Calinescu, David S. Johnson, Howard Karloff, Katrina Ligett, and Jia Wang. Compressing rectilinear pictures and minimizing access control lists. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1066–1075, 2007.

[3] F. Baboescu, S. Singh, and G. Varghese. Packet classification for core routers: Is there an alternative to cams. In *IEEE INFOCOM'03*, 2003.

[4] F. Baboescu and G. Varghese. Scalable packet classification. In *ACM SIGCOMM'01*, 2001.

[5] Beate Bollig and Ingo Wegener. Improving the variable ordering of obdds is np-complete. *IEEE Trans. Comput.*, 45(9):993–1002, 1996.

[6] E. Cohen and C. Lund. Packet classification in large isps: design and evaluation of decision tree classifiers. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 73–84, New York, NY, USA, 2005. ACM Press.

[7] Qunfeng Dong, Suman Banerjee, Jia Wang, and Dheeraj Agrawal. Wire speed packet classification without tcams: a few more registers (and a bit of logic) are enough. *SIGMETRICS Perform. Eval. Rev.*, 35(1):253–264, 2007.

[8] A. El-Atawy, K. Ibrahim, H. Hamed, and E. Al-Shaer. Policy segmentation for intelligent firewall testing. In *NPSec*, November 2005.

[9] A. El-Atawy, T. Samak, E. Al-Shaer, and H. Li. On using online traffic statistical matching for optimizing packet filtering performance. In *IEEE INFOCOM'07*, May 2007.

[10] A. Feldmann and S. Muthukrishnan. Tradeoffs for packet classification. In *IEEE INFOCOM'00*, March 2000.

[11] Errin W. Fulp. Optimization of network firewalls policies using directed acyclical graphs. In *Proceedings of the IEEE Internet Management Conference*, 2005.

[12] David Goldsmith and Michael Schiffman. Firewalking: A traceroute-like analysis of ip packet responses to determine gateway access control lists, http://www.packetfactory.net/firewalk/firewalk-final.html. White paper, Cambridge Technology Partners, October 1998.

[13] P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, 15(2):24–32, 2001.

[14] P. Gupta and N. McKeown. Packet classification using hierarchical intelligent cuttings. In *Interconnects VII*, August 1999.

[15] P. Gupta, B. Prabhakar, and S. Boyd. Near optimal routing lookups with bounded worst case performance. In *IEEE INFOCOM'00*, 2000.

[16] H. Hamed, A. El-Atawy, and E. Al-Shaer. Adaptive statistical optimization techniques for firewall packet filtering. In *IEEE INFOCOM'06*, April 2006.

[17] Hazem Hamed and Ehab Al-Shaer. Dynamic rule-ordering optimization for high-speed firewall filtering. In *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 332–342, New York, NY, USA, 2006. ACM.

[18] INTEL. Intel 64 and ia-32 architecture software developer's manual. "http://www.intel.com/design/processor/manuals/253666.pdf", 2007.

[19] J.R. Burch, E.M. Clarke, D.E. Long, K.L. MacMillan, and D.L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(4):401–424, 1994.

[20] L. Kencl and C. Schwarzer. Traffic-adaptive packet filtering of denial of service attacks. In *WOWMOM'06: The 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, pages 485–489, Washington, DC, USA, 2006. IEEE Computer Society.

[21] Karthik Lakshminarayanan, Anand Rangarajan, and Srinivasan Venkatachary. Algorithms for advanced packet classification with ternary cams. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 193–204, New York, NY, USA, 2005. ACM.

[22] A. J. McAulay and P. Francis. Fast routing table lookup using CAMs. In *IEEE INFOCOM'93*, March 1993.

[23] Derek Pao, Yiu Keung Lia, and Peng Zhoua. Efficient packet classification using tcams. *Computer Networks*, 50(18):3523–3535, Dec 2006.

[24] T. Samak, A. El-Atawy, E. Al-Shaer, and H. Li. In *IEEE ICNP'07*, October 2007.

[25] V. Srinivasan, Subhash Suri, and George Varghese. Packet classification using tuple space search. In *Computer ACM SIGCOMM Communication Review*, pages 135–146, October 1999.

[26] Thomas Y. C. Woo. A modular approach to packet classification: Algorithms and results. In *IEEE INFOCOM'00*, pages 1213–1222, March 2000.

[27] M. Yoon, S. Chen, and Z. Zhang. Reducing the size of rule set in a firewall. In *ICC'07: Proceedings of the IEEE International Conference on Communications*, pages 1274–1279, June 2007.