

# Using Online Traffic Statistical Matching for Optimizing Packet Filtering Performance

Adel El-Atawy, Taghrid Samak, Ehab Al-Shaer  
School of Computer Science, DePaul University  
Chicago, USA  
{aelatawy,taghrid,ehab}@cs.depaul.edu

Hong Li  
Intel  
hong.c.li@intel.com

**Abstract**—Packet classification plays a critical role in many of the current networking technologies, and efficient yet lightweight packet classification techniques are highly crucial for their successful deployment. Most of the current packet classification techniques exploit the characteristics of classification policies, without considering the traffic behavior in optimizing their search data structures. In this paper, we present novel techniques that utilize traffic characteristics coupled with careful analysis of the policy to obtain adaptive methods that can accommodate varying traffic statistics while maintaining a high throughput. The first technique uses segmentation of the traffic space to achieve disjoint subsets of traffic properties and build bounded depth Huffman trees using the statistics collected for these segments. The second technique simplifies the structure maintenance by keeping the segments ordered in a most-recently-used (MRU) list instead of a tree. The techniques are evaluated and their performance are compared. Moreover, attacks targeting the firewall performance are discussed and corresponding protection schemes are presented.

## I. INTRODUCTION

Packet classification is a critical component that determines the performance of many network devices; including firewalls, IPSec gateways, Intrusion Detection Systems, DiffServ and QoS routers. The main task of packet filters (or classifiers) is to categorize packets based on a set of rules representing the filtering policy. The information used for classifying packets is usually contained in the header fields in the packet, which are (for a simple firewall) protocol field, source IP/port, and destination IP/port in IPv4. Each filtering rule  $R$  is a tuple of field values. A packet  $P$  is said to match a rule  $R$  if all header-fields of  $P$  match all corresponding rule-fields of  $R$ . In firewalls, each rule  $R$  is associated with an action (*i.e.*, category to place the packet) to be performed if a packet matches a rule. These actions indicate whether to block (“deny”) or forward (“allow”) the packet to a particular interface. For example, a filtering rule  $R=(TCP, *, *, 140.192.*:23, allow)$  matches traffic destined to subnet 140.192 and TCP destination port 23 only regardless of the sender. A packet classification policy consists of  $N$  rules  $R_1, \dots, R_N$ . Since any packet may match multiple rules in the policy, based on the rule ordering, the first matching rule is given the highest priority. If a packet does not match any of the rules in the policy, then it is discarded because the default rule (implicit last rule) is assumed to be deny [2]. For other security devices than firewalls, the fields used for packet classification can include

many other fields, some might depend on the packet payload. For example, an IDS might further classify packets depending on the ftp command mentioned in the packet or whether the http packet contains a PUT or a GET command. Thus, it is highly important to have a technique that can scale well with increasing the number of fields on which the classification decision takes place.

Moreover, with the dramatic advances in the network/link speed, packet filtering/classification techniques must be constantly optimized to cope with the network traffic demands and attacks. This requires reducing the packet matching time needed to “allow” and “deny” packets. This problem is even more critical when application-level filtering is used. Thus, efficient yet easy to implement packet filtering techniques are highly crucial for successful deployment of traffic filtering/classification technologies on the Internet.

This work is highly motivated by the Internet traffic properties that were observed in our study in this paper and previous results in [16] and [14]. Our study of many Internet and private traces shows that the major portion of the network traffic/flows matches a small subset of the firewall rules. In other words, the frequency distribution for some of the traffic properties appears to be highly skewed. We also observed that this “skewness” in traffic distribution is likely to stay for time intervals that are sufficient to make such skewness important to consider in packet filtering. Therefore, we propose techniques that can utilize this phenomenon.

The first technique is based on Huffman trees. The tree is built using traffic space segments for symbols. The idea of segmenting the traffic address space was used before in testing firewall implementations [8]. Basically, all segments are disjoint subsets of the traffic space such that each subset contains flows that have some common characteristics. The second technique uses segments as well but exploits the fact that in some cases the locality of the traffic is high enough that just a few segments cover all of the traffic investigated. It keeps a list of segments sorted on a most-recently-used (MRU) basis. By this, the cost of periodical maintenance is eliminated.

Packet filtering optimization has been studied extensively in the research literature [10]. However, most of the current packet classification techniques exploit the characteristics of filtering rules but they do not consider the traffic behavior in their optimization schemes. Being deterministic, these tech-

niques guarantee an upper bound on the packet matching time. On the other hand, our statistical matching approaches aim to improve the average filtering time with a limited bound on the worst case. In addition, unlike many of the presented techniques, our techniques have much less space complexity. The use of statistical trees for optimizing packet classification was discussed in [5], [12], [14]. However, in [12], the technique was limited to only one field (routing prefix) that has a given frequency distribution and in [14], the technique was not able to scale with the number of fields if to be used for IDSs. Also in [5], the technique used whole rules without exploiting the traffic patterns over separate fields. In this paper, our schemes - on the other hand - use statistical trees or dynamic lists over segments that provide much finer granularity than whole rules and are dynamically updated to reflect the overall network traffic statistics with the added capability to digest multiple fields.

The paper is organized as follows. In Section II we describe the previously published related work. We show a brief study of network traffic statistical characteristics in section III. In sections IV, and V we present the two proposed techniques. We then present some attacks and defence techniques in section VI. Section VII shows the evaluation study for the proposed techniques. Finally, in Section VIII we present the conclusions and our plans for future work.

## II. RELATED WORK

The packet classification problem has been extensively studied. The simplest form of a packet filter algorithm sequentially searches the rule list until a match is found. The scalability of this solution is generally poor, as the search time is proportional to the policy size. The main solutions to improve the search times use various combinations of one or more of the following: hardware-based solutions, specialized data structures, and heuristics.

HW based solutions using Content Addressable Memories (CAM) exploit the parallelism in HW to match multiple rules in parallel. They are limited to small policies because of cost, power and size limitations of CAMs. Other hardware based solutions are described in [18]. The policy rules are structured as a trie, with a linear classification time in the number of bits used in the criteria fields. This value can still be exceedingly large (*e.g.*, for the basic 5-tuples,  $B = 104$ ). Aggregated Bit Vector (ABV) in [3] solves the problem with separate lookups for each dimension, followed by a combining phase. For each dimension, a lookup is done using a trie that returns a list of all matching rules. The final result is then computed by finding the rule with highest priority. To save space, this approach uses compressed bit vectors for rule lists.

A wide variety of specialized data structures have been used for fast packet classification. Srinivasan et al. [20] build a table of all possible field value combinations (cross-products), and search it quickly by doing separate lookups on each field, then combining the results into a cross-product table followed by indexing into the table. However, the size of the cross-product table grows dramatically with the number of rules.

A geometric algorithm was proposed by Feldmann et al. [9], introducing a data structure called Fat Inverted Segment (FIS) Tree. The first dimension is partitioned with the endpoints of the projection of the rules on that dimension. Each of the segments is then partitioned, along the remaining dimensions, into a number of  $d$  dimensional regions. To avoid an explosion of the storage requirements, the  $d$  dimensional regions are linked in a FIS tree of bounded depth, and the common partitions are pushed up in the FIS tree. The main advantage of the FIS technique is that it scales well with the number of filtering rules.

Work on decision-tree based classification algorithms using geometric cutting was introduced by Gupta and McKeown [10] and Woo [22]. Both build a decision tree using local optimization decisions at each node to choose the next bit or field to test. The paper by Woo [22] goes one step ahead by using multiple decision trees. This step increases search time but it can greatly reduce storage. Similarly, the Hierarchical Cuttings (HiCuts) scheme described in [11] uses range checks instead of bit tests at each node of the decision tree. In [5], decision trees were also used with common-branch adjustment to reduce space requirements. However, the elements in the decision trees are the whole rules, and the choices used to build the tree are greedy choices.

Gupta and McKeown [10] proposed a heuristic approach called Recursive Flow Classification (RFC). One advantage of RFC is that the various lookup stages can be pipelined, so in a hardware implementation the classifier can have a very high throughput. However, this approach does not scale easily to medium or large number of rules.

Although all previous work contribute significantly to the advancement of packet classification research, their main objective was to improve the worst-case matching performance. Hence, they do not exploit the statistical filtering schemes to improve the average packet matching time. In addition, they mostly exhibit high space complexity. However, in [14] a different approach was taken that targets the traffic that will eventually hit the default rule after a mismatch with every single rule in the policy. The technique uses new statistics-induced rules that can eliminate as much as possible of such traffic with minimal overhead to other flows. Also Lukas [15] and Znati introduced some steps to make search structures adapt to traffic dynamics.

The related work closest to our approach are those proposed by Gupta [12] and Hamed [14]. 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. The authors show that using statistical data structures can significantly improve the average-case lookup time. As the focus of this paper is on routing lookup, the scheme is limited on search trees of a single field with arbitrary statistics. In the second paper, the work was extended to multiple fields in firewall policies with the capability of parallel processing.

### III. TRAFFIC ANALYSIS

One of the important traffic characteristics commonly observed in our analysis of large number of Internet and private traces is the skewness of the traffic matching in the policy, in the sense that the majority of the inbound (or outbound) packets match against a small subset of all filtering criteria that exists in the policy. The two main characteristics in internet traffic that makes our approaches very promising are that (1) traffic skewness is always clearly evident and that (2) it is unlikely to change over a short period of time. Thus, utilizing these properties will yield techniques that are very efficient on a packet-per-packet basis, as well as they will have a very low maintenance cost.

In this section, we represent the definition of the “segment” and highlight some observations and properties of Internet traffic under the light of this definition. The traffic analysis was performed on several Internet packet traces collected at the edge routers of DePaul University and University of Auckland networks [19]. The traces are stored as one-hour packet header logs at different days of week and times of day, each containing the header information for 3M to 10M packets that reflect realistic network conditions.

#### A. Traffic Space Segmentation

Here we discuss briefly the segmentation concept [8].

*Definition 1: Traffic Space:* Is the space whose elements are all of the possible packet-information fields tuples.

For example, for basic IP4-based firewall rules that consists of only  $\langle \text{protocol}, \text{srcIP/port}, \text{dstIP/port} \rangle$ , we have a space whose size is  $2^{8+32+16+32+16} = 2^{104}$ .

*Definition 2: A Segment:* Is the subspace (*i.e.*, subset of the overall traffic space) whose elements match the same and exact set of rules from the policy.

The process of converting a list of rules into a set of segments is quite straightforward. We iteratively calculate the different overlaps of all rules. Starting with the first two rules, intersecting them results in (a maximum of) four segments. These four segments will each be intersected with the third rule, and so on. Empty segments (*i.e.*, from intersecting two disjoint rules or segments) are removed once found to avoid exponential explosion in the number of segments. While creating segments from overlapping rules, the rule-order is considered to identify the action (*i.e.*, permit/deny) of the resulting segments. The space of any given segment  $S_i$  is represented by a Boolean expression  $Expr(S_i)$  that represents the segment’s area (as a conjunction of each of the original rules as well). Such an expression will evaluate to true if the bit-values of a packet header that belongs to this segment is used. For example, assume having a simple packet header that consists of only source and destination fields each is just 3-bits long. If we are given the two rules in Table I-a for a simple environment, the segmentation will be as in Table I-b.

#### B. Packet flow properties

Studying the statistics of various Internet traffic traces, we observed a number of properties for Internet flows. From our

TABLE I

EXAMPLE FOR THE PROCESS OF SEGMENTATION

Rule	src	dst	expression
$R_1$	11*	0**	$x_0x_1\bar{x}_3$
$R_2$	1**	01*	$x_0\bar{x}_3x_4$

(a) Sample Rules

Segment	Origin	Expression
$S_1$	$R_1 \wedge R_2$	$x_0x_1\bar{x}_3x_4$
$S_2$	$\bar{R}_1 \wedge \bar{R}_2$	$x_0x_1x_3x_4$
$S_3$	$\bar{R}_1 \wedge R_2$	$x_0\bar{x}_1x_3x_4$
$S_4$	$R_1 \wedge \bar{R}_2$	$\bar{x}_0 \vee x_3 \vee \bar{x}_1x_4$

(b) Segments

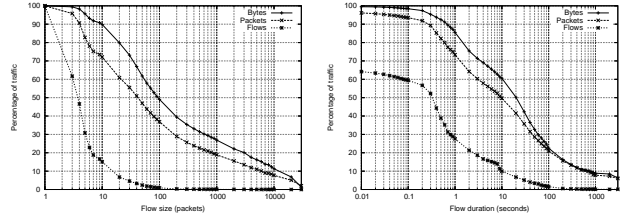


Fig. 1. CCDF distribution of (a) flow size, and (b) flow duration for Internet traces at a University boundary

analysis, Figure 1-(a) shows that about 60% of the flows have 3 packets or less, while 20% have 10 packets or more. The figure also shows that the long flows carry around 70% of the Internet traffic. Similarly, Figure 1-(b) shows about 50% of the flows last 2 seconds or less, while 20% last 10 seconds or more. Thus, long-lived flows carry most of the traffic. Thus, these observations indicate that a small portion of the firewall policy (rules) is used for matching a significant portion of the traffic packets over a considerable amount of time. Previous studies [16] have also shown that while the majority of Internet flows have short flow sizes, the considerable amount of Internet traffic is constituted from the long flows. A similar observation was also shown for the flow duration. As a result, this shows that filtering optimization based on packet frequency is not only useful for improving the overall matching performance but also practical in most cases.

#### C. Statistical measures

To analyze the policy and the traffic we use two metrics. The first is the skewness of the traffic frequency over the policy segments. By monitoring the number of packets that belong to each segment we build a frequency distribution over the domain of segments. The skewness factor  $S$  is a value between 0 (for a uniform distribution) and 1 (for a totally skewed distribution). To calculate  $S$ , we use information theory formulation to quantify the Entropy of a given distribution [6].  $S$  is defined by the formula:

$$S = 1 - \frac{\sum_{i=1}^n p_i \lg p_i}{\lg n} \quad (1)$$

where  $p_i$  is the probability of segment  $i$  and it is calculated as the ratio of the number of packets matching segment  $i$  to the total number of packets received. Also  $n$  is the total number of segments.

The other metric is the correlation between the frequency distributions obtained in successive time windows. This metric indicates the stability of the frequency distribution over time. If the correlation is close to unity then conclusions about the traffic from one time window will be valid in the next window. The segment frequency distribution is said to be *time-correlated* if the frequencies of a segment is similar over the two intervals. We use the *correlation factor C* as a value between 0 (uncorrelated distribution) and 1 (totally-correlated distribution), and it is calculated as follows [7]:

$$C = \frac{\sum_{i=1}^n (p_i - \mu_p)(q_i - \mu_q)}{n \cdot \sigma_p \cdot \sigma_q} \quad (2)$$

where  $p_i$  is the probability of segment  $i$  in a time interval, and  $q_i$  is the probability in the following interval. The quantities  $\mu_p$  and  $\mu_q$  represent the mean, while  $\sigma_p$  and  $\sigma_q$  represent the standard deviation of the probability distributions.

#### D. Segment statistical properties

Now we will analyze the distribution of traffic packets over policy segments. The traces used for the analysis are the same that were originally used in [14], and the policies are policies relevant to the traffic capture point. Number of hits for every segment are recorded, and the skewness calculated every specific window of time. The traces are analyzed using different window sizes. The results are in Figure 2. It shows a considerable skewness level for all policies tested. As expected, when the policy increases in size, the skewness goes even higher. Another very important observation is that the decision on the optimal window size is not a major issue for techniques based on segments.

To measure the stability of this skewness, we measure the correlation between the frequency distribution of successive time windows (Figure 3). The results are much more impressive than their counterparts of field values obtained in [14]. This shows that it is very safe to build structures that can last for considerable time without being outdated.

### IV. SEGMENTS-BASED TREE SEARCH (STS)

Using the ideas of Huffman tree, and the segmentation of traffic address space, it is possible to minimize the average number of comparisons applied on packets arriving at the firewall ports. The idea is grounded in the traffic space segmentation technique, and the obvious skewness of the amount of traffic belonging to each of the segments. Using the fact that these areas are non-overlapping, renders it possible for constructing a search structure that is efficient and responsive to traffic statistics without being constrained with order or relation between segments.

#### A. Collection of Statistics

For this technique, statistics will be kept as hit-count measurements for each of the segments. Once a decision is reached that a packet falls within a specific segment, this segment's counter will be incremented. While this is a simple operation, the technique presented in the following section (SLS) eliminates the cost of frequency counting.

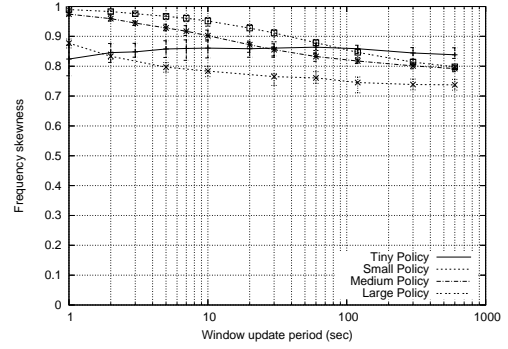


Fig. 2. Skewness of the packet frequency distribution over policy segments. Each plot shows the average and error bars indicate the minimum and maximum skewness levels obtained using this window size. Policy sizes are indicated in Fig. 4



Fig. 3. Correlation between packet frequency distribution (over policy segments) between successive time window. Each plot line shows the average and error bars indicate the minimum and maximum correlation levels obtained using this window size. Policy sizes are indicated in Fig. 4

#### B. Building the Tree

Given the policy in the form of a list of rules, segmentation will be performed to result in a list of segments, each associated with an action and a Boolean expression representing its space [8]. It is required to have a search structure that uses these segments to find the suitable action for each of the incoming packets in the minimum number of comparisons in the average sense. So, we redefine the measure of “comparisons” to fit our technique of segment-membership checking to be as follows:

*Definition 3: Comparison:* An evaluation of a Boolean expression using the packet header values.

We will use the method used for building depth constrained Huffman trees [17] to build a search tree to attain the goal with bounded worst case. After obtaining the segments list, each segment is considered a symbol into the Huffman algorithm. A segment has the following properties: an action (obtained from the top-most rule intersecting at this segment), a Boolean expression representing its space, and a Weight (initially all segments have the same weight, but as statistics are collected these weights will be taken proportional to the hits in the segment). The output tree of the Huffman algorithm (with the additional depth constraint) is our new comparison tree. From [4], constraining the worst case to be not more than two levels over the average will yield an overall increase in the

average search time by no more than 0.5 comparisons/packet. This is considered a very reasonable overhead in order to guarantee a reasonable worst case.

1) *Segments Simplification*: As an optimization step, we simplify the segment list for smaller search tree (*i.e.*, shorter average depth). By iterating through segments with the same action, we can see if there are segments that can be merged (*i.e.*, ORing their Boolean expressions) together in order to simplify their common expression. This step is an off-line step that needs to be performed once for the policy along with the segmentation itself. However, performing the simplification can yield to dilution of the skewness of the segment weights which in turn can lower the effectiveness of the whole technique. Therefore, instead of blindly applying the simplification phase, or applying it at run-time and suffer from the computational overhead; we propose using an empirical threshold on its applicability. If the simplification lowered the number of segments below this threshold then we can assume the savings from a smaller tree will overcome the flatness that might occur in the frequency distribution. Reductions in the segment list length by 10-20% is typical. However, as these results depend solely on the structure of the policy, this additional step is disabled in the test cases discussed in the evaluation for objective results (See Section VII).

2) *Tree Building Algorithm*: Some modifications to the original Huffman’s algorithm implementation need to be performed; symbols used for leaves have a more complex structure and comparisons are not as simple as what is originally used in traditional Huffman tree (See Algorithm 1):

- The internal nodes contain a Boolean expression that should be satisfied by each packet passing through this node towards its descendants. The expression is built by ORing the two corresponding expressions from the children nodes (Algorithm 1, line 10). As we build the tree, when a new node is created in a merging step of two nodes with the lowest weight, we “OR” the expression of these two nodes and place it in the newly created node.
- The decision expression by which we choose one of the two child paths to go through is made by using the simpler (*i.e.*, easier to evaluate) expression of the two children’s expressions ( Algorithm 1, line 6-8). If the evaluation results in a satisfying assignment we go with this node, otherwise, we go to the other direction. This is in contrast to the simpler bit comparison that is performed in the decoding process using Huffman codes. However, this can also be modeled as checking a Boolean expression with a single variable. This is just a generalization of the original branching decision condition and reserve all the characteristics regarding complexity of search operations.
- The height of the tree and the size of the incoming data to be matched against the tree are not related. In contrast with the Huffman tree when used in decoding variable sized symbols, this tree is used to reach a decision based on the information contained in a fixed size incoming packet but using different decision criteria.

---

**Algorithm 1** Building the Segments-based Huffman Tree: *HuffmanBuild (SEGLIST)*

---

```

1: MinHeap  $H \leftarrow SEGLIST$  {order on weights}
2:  $n \leftarrow |SEGLIST|$ 
3: while  $n \geq 2$  do
4:    $H \rightarrow s_1$ 
5:    $H \rightarrow s_2$ 
6:   if  $Expr(s_1)$  longer than  $Expr(s_2)$  then
7:      $Swap(s_1, s_2)$ 
8:   end if
9:   new Node  $s_3$ 
10:   $Expr(s_3) = Expr(s_1) \vee Expr(s_2)$ 
11:   $s_3.left = s_1$ 
12:   $s_3.right = s_2$ 
13:   $wt(s_3) = wt(s_1) + wt(s_2)$ 
14:   $H \leftarrow s_3$ 
15:   $n \leftarrow n - 1$ 
16: end while
17:  $H \rightarrow s_{root}$ 
18: return  $s_{root}$ 

```

---



---

**Algorithm 2** Packet Classification using a Segment-based Huffman Tree: *HuffmanFilter (packet)*

---

```

1:  $node \leftarrow root\_node$ 
2: while  $node$  is not leaf do
3:   decision = Evaluate ( $Expr(node.left)$ , packet)
4:   if decision = true then
5:      $node = node.left$ 
6:   else
7:      $node = node.right$ 
8:   end if
9: end while
10: increment  $node.frequency$ 
11: return  $node.Action$ ;

```

---

As we can see in Algorithm 1, the operation is a slight modification of the original Huffman tree building algorithm. The changes occur in swapping the left and right children based on whichever is easier to evaluate. Another difference is the added expression at each node. Normal operations stay the same; choose the pair of nodes having lowest combined weight, remove them and add them as children to a newly created node. This node has the weight equals the combined weight of the two nodes, and its expression equals the ORED of the two expressions of the combined nodes.

### C. Packet Filtering

When a packet is received, it is first decoded to extract the information bits used for filtering. At each node of the search tree, the packet is checked against the Boolean expression of the children of this node. The packet is directed to the node whose Boolean expression is satisfied. This process goes on till a leaf is reached. The action associated with this segment is applied to the packet, and the frequency of this segment is updated as well. Algorithm 2 shows the operation in an iterative way. A recursive form can be written but the iterative version saves the overhead of recursive calls and is very simple.

## V. SEGMENTS-BASED LIST SEARCH (SLS)

In the previous section, we showed how we can use the skewness of the segments to build a Huffman tree over these segments to enhance the performance of searching. However,

this technique have the overhead of periodical maintenance of the tree. To eliminate this cost, we introduce in this section another technique to utilize the very high imbalance of the frequency distribution of packets over the policy segments. By building a simple single-list of segments that is updated after each packet match, we obtain a very simple yet extremely efficient structure. Each packet is matched against the segments one-by-one until a match takes place. Once the matching segment is identified, we move it to the top of the list. This operation takes  $O(1)$  steps to manipulate the list pointers of the matching segment, the one before it, and the top of the list. A very simple operation that has the only downside of having a transient behavior till a good order of segments is obtained. See Section VII for a demonstration of this transient effect.

In the evaluation section, we see that the high skewness is very rewarding in this technique. Although the worst case is linear in the number of segments, most of the packets match one of the top few segments (typically one of the top ten segments). In contrast to our previous work in [13], the dependency between rules does not pose any problem. The use of segments instead of rules has this effect of separating dependency and making any order of segments acceptable.

## VI. ATTACKS AND DEFENSES

In this section we present a few ideas of possible threats that are tailored specifically for statistical packet filtering techniques. The common basic idea of these techniques is how to use the optimism of the filtering device (firewall, IDS, etc.) regarding the steadiness of the traffic frequency distributions to force it to have lower performance than anticipated. Clearly, this is a very incomplete coverage of possible attacks, but it shows how different attacks can be planned against the firewall.

### A. Attacks

1) *Rapid Statistics*: If the firewall links are very under-utilized; an attacker can drive the traffic statistics into the form she wishes. By injecting a not very high traffic volume, the attacker will be aware of the internal statistics collected by the firewall with a great accuracy. Once this information is available, the attacker can keep on sending the most unexpected traffic to the firewall (*i.e.*, the traffic whose parameters are statistically farthest from previous window). Traffic that was thought to be rare will face longer than average filtering time, causing the overall performance to drop well below the average. SLS will suffer the most from this attack due to the fact that its worst case is much higher than its average.

2) *Sudden Visitors*: In this attack, the adversary has very little to do regarding the overall traffic characteristics. This might be the normal case for busy firewalls, with steady massive traffic. The only advantage for the attacker is that the traffic characteristics are somehow static, or dynamic with slow rate of change; which means that he might be aware to some extent of the natural/normal traffic behavior. The approach for the attacker is to use as much as he can from his

own bandwidth to inject to the firewall a flow with the least expected properties (*e.g.*, very unusual/invalid protocol, unused service, a normal workstation as a destination, an illogical combination of packet header field values, etc). These packets will have a much longer service time at the filtering module than the average packet. Thus, the firewall's packet buffer will tend to get filled bit by bit as the attacker injects such hard to process packets, eventually causing overflow and packets to be dropped. Again SLS will be more affected by this attack as one bad packet will increase the search time for all successive distinct packets by one.

### B. Defenses

1) *Dynamic Window Update*: The size of the time window at which we update the search structure in the first technique (STS) is crucial to its performance. Choosing the best window size depends on both the traffic statistics and the policy structure and size. The traffic statistics and volume tend to change in a periodic fashion due to human activity patterns and many other factors. The change due to this relatively long cycles is not the main concern for a firewall filtering technique. The main factor we are concerned with is the short term variations that occur due to individual flows or user groups activities. More importantly malicious traffic; like tailored malicious flows, and DDoS attacks. For these cases (and other normal cases that might look like malicious traffic statistically), we have to dynamically change the window size to stay on top of these activities. In Fig. 2 and 3, we can see the effect of increasing the window size in the small range: one second up to about 10 minutes. We see that it is safe to change the window size without any harsh effect on the statistics observed in a healthy environment.

We propose having an open window size that expires only when the performance degrades below a certain threshold. By keeping track of the average number of comparison performed for every packet, we can tell when the filtering performance is degrading. Once the average cost/packet rises above a predefined threshold, the search structure (the Huffman tree) is rebuilt. The *optimization efficacy*  $\varepsilon_f$  is defined as the actual reduction in matching as compared to binary search when the current traffic is matched against a search tree built using the traffic statistics collected in the previous time interval. Mathematically,  $\varepsilon_f$  is given by the following formula:

$$\varepsilon_f = 1 - \frac{\sum_{i=1}^n q_i \lg p_i}{\lg n} \quad (3)$$

where  $q_i$  is the probability of entity  $v_i$  (segment or field-value) in the current time interval, and  $p_i$  is the probability of this entity in the preceding interval.

Although this formula accurately estimates the matching gain, it is quite expensive to be performed at runtime for every packet. A lightweight approximation that gives very close average results is the exponential moving average of the

matching gain  $\bar{\varepsilon}$ .

$$\bar{\varepsilon}_i = (1 - \omega)\bar{\varepsilon}_{i-1} + \omega g_i \quad \text{where } g_i = \frac{h_i - \lg n}{\lg n}$$

$$\bar{\varepsilon}_i = (1 - \omega)\bar{\varepsilon}_{i-1} + \left(\frac{\omega}{\lg n}\right)(h_i - \lg n) \quad (4)$$

$$\bar{\varepsilon}_{thr} = \tau \varepsilon_{opt} \quad (5)$$

where  $h_i$  is the height (*i.e.*, number of comparisons) of the destination leaf (*i.e.*, segment) of packet  $i$ ,  $g_i$  is the gain over binary search for packet  $i$ . After a packet is matched using the tree, if  $\bar{\varepsilon}_i$  drops below a certain threshold  $\bar{\varepsilon}_{thr}$ , the search tree is disposed and a new tree is built.  $\bar{\varepsilon}_{thr}$  is calculated as a ratio  $\tau$  of the optimal gain  $\varepsilon_{opt}$  that the tree was built to achieve. Notice that these expressions involve only inexpensive addition and multiplication operations.

To avoid long periods of mediocre performance that is just above the rebuilding threshold, a periodic update is performed every constant (and relatively long) intervals of time. Using the latest traffic statistics, a new matching tree is constructed using fresh statistics to boost back the matching performance close to its optimum level. Since this type of update is mandatory, the update period should be based on the computational capacity of the filtering device. A reasonable update period can be as high as one hour.

Under an attack, the period after which the tree is rebuilt can be dangerously short, overloading the processing power of the firewall by rebuilding operations. Thus, once a tree is found to last for less than some safety threshold, the rebuilding threshold should be relaxed (temporarily) and if the problem persists for a few times, then the statistical approach should be changed. One can switch to a less demanding technique as SLS, for the no-maintenance overhead even if it is has a higher average cost. After such protective switch, the tree should become active again, and operation continues as before the attack. Of course, this can be the other way around with STS substituting SLS.

2) *Dynamic Sampling Rate*: As a rule of thumb, it is hard to keep track of each and every packet, and we have to perform sampled study of the traffic targeting the firewall. Sampling rate has to be chosen according to many factors, especially the available processing power of our device. However, sampling rate can be chosen to specifically mask out burst attacks. By lowering sampling rate, we diminish the effect of burst attacks in changing our overall statistics. The attacker has to maintain a high rate of transmission for a long time to make a considerable difference in our statistics, and over this long time our firewall will be safe and not suffering extra pressure due to the attack. Although our system will not be able to ideally follow up with the high dynamics of the traffic, this is not a very high price to pay to avoid complete loss of performance due to the erroneous statistics that would - otherwise - be fed into the technique. Moreover, the dynamic window update should take care of triggering a rebuild up of our structure using new statistics if filtering performance dropped considerably.

3) *Split Load and Parallel Searching*: The idea of this defense technique is to remove the competition over CPU resources between frequent and non-frequent packets. Traffic is split over a set of processors in a way that the traffic with similar popularity (*i.e.*, popular segments vs unpopular ones) be assigned to the same processor. Also, each processor would have its own private queue. Packets are forwarded to their designated queue by a separate processor that is responsible for -periodically- finding a reasonable partition among segments such that the decision to be taken later for each packet be as simple as possible (*i.e.*, a single term Boolean expression which is the ORing of all the segments in the same bin). After each window of time the partitioning processor will decide on the previous statistics where it can place the split.

For example, if we have two processor system, by evaluating a single expression, we can know which processor is responsible for an incoming packet. Such a scheme means that the split is made over the Zipf distribution of the segments frequencies. Using an approximation for the solution of this partitioning problem, we can reach a reasonable decision in a practical timely manner.

Another way of simplifying the search structure and utilizing the power of parallel processors is by running the Huffman algorithm till we have a forest instead of going all the way to forming a single tree. Each processor will be responsible for one of the trees. The average search time will be still optimized. This algorithm and its performance were discussed theoretically in [1].

## VII. PERFORMANCE EVALUATION

In this section, we show the differences between the presented techniques and study their performance in different situations.

*Overall Performance of filtering*: The number of operations performed per packet is the main figure to observe and analyze the performance of filtering techniques. The average and minimum/maximum values of this metric are both used to make sure that - in average - the system will perform as anticipated and it is very rare that performance will drop below a dangerous level with the varying traffic statistics.

For STS, we see in Fig. 4 that the performance gain was in a practically attractive range (60-80%). In Fig. 5, the average number of operations per packet is shown. The average is taken over the whole packet trace, with the tree updated every indicated time window in the abscissa. Taking into consideration that the tree depth can be bounded, we have a very attractive filtering technique with less than five evaluations for every packet (including the depth constraint). Comparing this to the alphabet tree used in [14], the cascaded alphabet tree structure was not build to guarantee a worst case, and the presence of multiple fields made sure that the filtering algorithm will use at least one comparison per field. This makes their average case  $\sum_{i=1}^d H(F_i)$ , where  $d$  is the number of fields and  $H(F_i)$  is the entropy of the value statistics of field  $i$ . This figure easily sums up to much higher than the one we show in Fig. 5.

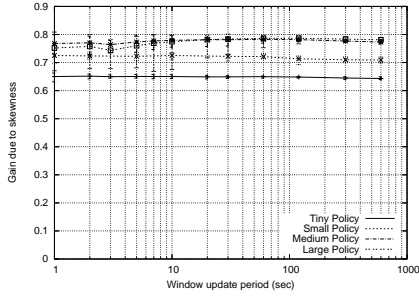


Fig. 4. Gain in filtering time by applying traffic statistics to segments search tree (STS) for policies of different sizes: 23, 138, 531, 934 rules and 68, 807, 6993, 21849 segments respectively.

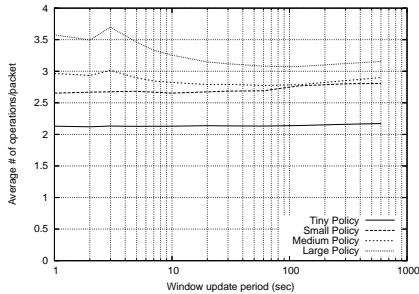


Fig. 5. Actual cost per packet in number of evaluations using STS. For policies of different sizes as in figure 4.

For SLS, we see in Fig. 6 that the cost is even lower than the average cost achieved in the case of STS. However, due to the fact that some packets might face much longer list of evaluations, we suggest that SLS be used only when traffic is known to be, more or less, in a steady state where SLS will surely excel over STS. Another factor is that if the firewall or IDS is close to its full computational capacity, then STS will be safer to use if the traffic behavior is highly dynamic.

In Fig. 7 and Fig. 8, only a short duration at the beginning of the SLS execution is displayed to show the transient effect until a steady state is obtained. We can see that most packets fall within the top 10 popular segments. This is a great motivation for using hardware support for these top segments, in order to have faster evaluation for their expressions.

Comparing these results to the technique of Dynamic Rule Ordering (DRO) introduced in [13], we can see that the stability of the popular rules, and the size of the active set makes our new technique superior in performance and scalability.

The evaluation was performed using both real-life policies as well as automatically generated policies with structures and statistics following the results obtained by other researchers [21] and [23]. The traces were taken from the archives available from NLANR [19].

As inter-rule interaction increase, the number of segments increases (see Figure 9). However, an exact break even point between DRO and Segments-based filtering is hard to know in advance because of the huge number of parameters involved.

*Resistance to attacks:* STS is more robust towards attacks in the aspect that it does not loose much performance as the

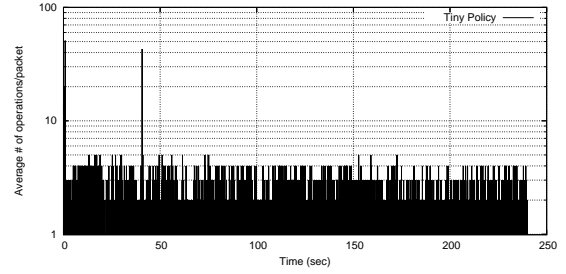


Fig. 6. Actual cost per packet in number of evaluations using SLS. For a small policy.

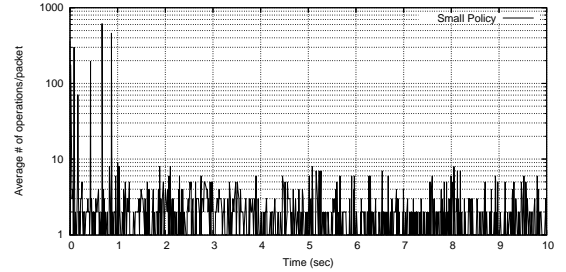


Fig. 7. Actual cost per packet in number of evaluations using SLS. Only first 10 seconds are shown to emphasize transient.

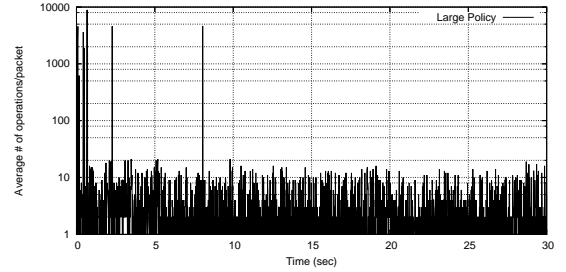


Fig. 8. Actual cost per packet in number of evaluations using SLS. Only first 30 seconds are shown to emphasize transient.

bounded-depth tree enforces a worst case. However, for the other technique the performance can have a significant drop if traffic was engineered for that purpose and precautions were not considered (as in Section VI).

*Preprocessing Time Complexity:* The segmentation process was shown to be  $O(ns)$  where  $n$  is the number of rules, and  $s$  is the number of segments resulting [8]. The Huffman tree building time is the standard  $O(n \lg n)$ .

*Periodic maintenance time overhead:* The overhead of periodically updating the search structure can be kept virtually unnoticeable by parallelizing its processing with the normal filtering operations. However, if this is not possible due to hardware restrictions then it has to be kept at a minimum not to excessively delay the filtering process. However, our study of various policies showed that the number of segments approximately grows linearly with the number of rules in the policy, as can be seen in Figure 9. For the second technique, there is no periodical maintenance required.

*Space Complexity:* For both techniques the overhead is the storage required for the segments themselves, no significant additional storage is required. The number of segments as shown in Fig. 9 does not grow dramatically with policy size.

*Potential of Parallelism:* As discussed in Section IV, the Huffman tree can be built, and searched in parallel. For SLS,

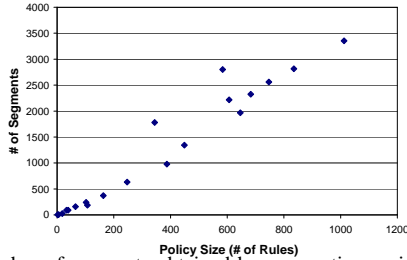


Fig. 9. Number of segments obtained by segmenting various policies

TABLE II

SUMMARY OF SPACE AND TIME COMPLEXITY OF THE TWO TECHNIQUES, COMPARED WITH DRO [13] AND ALPHABETIC TREE TECHNIQUE [14].

	STS	SLS	Alph_tree	DRO
Space Complexity	$O(s)$	$O(s)$	$O(d.n)$	$O(n)$
Time Complexity:				
Preprocessing	$O(n.s)$	$O(n.s)$	$O(n \lg n)$	$O(n \lg n)$
Filtering	$O(\lg s)$	$O(s)^*$	$O(\lg n)$	$O(n)$
Maintenance	$O(s \lg s)$	N/A	$O(n \lg n)$	$O(n \lg n)$

\* Note that filtering time for SLS is linear in worst case, but the average case is much lower as evident from the evaluation graphs

the list can be searched by multiple processors in parallel, but the update will pose a concurrency problem that is not easy to solve. This is part of the future extensions that are planned for this work.

## VIII. CONCLUSION

In this paper, two techniques are presented that can be used as packet filtering algorithms for firewalls, IDSs, etc., namely Segments-based Huffman Trees Search (STS), and Segments-based List Search (SLS). The techniques were designed to be light weight while utilizing the dynamic statistics of the traffic. However, it has been shown that each technique has its own strengths and the specific circumstances where each will excel. Attacks that are designed specifically for statistical filtering techniques are presented. Countermeasures for these attacks are discussed and their effectiveness investigated, and were shown to be effective in a sense that the attacker will not reduce the firewall's performance below a reasonable adjustable level. Performance evaluation of these techniques supports our claim that using the traffic statistics while designing a packet filtering engine can significantly reduce the average processing time. Results show that for each technique, the performance increase by using traffic statistics was typically in the 50-80% range. Gain as high as 90% was obtained in some cases, specially for the SLS technique. Moreover, no frequent maintenance operations are needed for STS, and none for SLS. As a conclusion, we claim that we can safely use traffic statistics-based techniques for packet classifications/filtering with no concerns regarding performance or security. Compared to previous techniques [13], [14], the proposed techniques are more scalable to higher dimensionality and policy size.

Many point needs to be further investigated to enhance performance as well as to prove theoretical bounds on the gain from the proposed techniques. Study of the nature of

policies and its effect on the potential of each filtering technique will prove very useful in many practical as well as theoretical venues. For hybrid deployment of these techniques, decision parameters on which a technique can be selected are very crucial for having a complete solution. Different search structures using traffic space segments can be studied for future enhancements.

## ACKNOWLEDGMENT

This research was supported in part by Intel IT. Any opinions, findings, conclusions or recommendations stated in this material are those of the authors and do not necessarily reflect the views of the funding sources.

## REFERENCES

- [1] J. Abrahams. Parallelized huffman and hu-tucker searching. *IEEE Transactions on Information Theory*, 40(2):508–, 1994.
- [2] E. Al-Shaer and H. Hamed. Discovery of policy anomalies in distributed firewalls. In *IEEE INFOCOM'04*, March 2004.
- [3] F. Baboescu and G. Varghese. Scalable packet classification. In *ACM SIGCOMM'01*, 2001.
- [4] D. Baron and A. Singer. On the cost of worst case coding length constraints. *IEEE Trans. Inf. Theory*, 47(7):3088–3090, 2001.
- [5] 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.
- [6] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & sons, 1991.
- [7] A. L. Edwards. *An Introduction to Linear Regression and Correlation*. W. H. Freeman and Co, San Francisco, 1993.
- [8] A. El-Atawy, K. Ibrahim, H. Hamed, and E. Al-Shaer. Policy segmentation for intelligent firewall testing. In *NPSec*, November 2005.
- [9] A. Feldmann and S. Muthukrishnan. Tradeoffs for packet classification. In *IEEE INFOCOM'00*, March 2000.
- [10] P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, 15(2):24–32, 2001.
- [11] P. Gupta and N. McKeown. Packet classification using hierarchical intelligent cuttings. In *Interconnects VII*, August 1999.
- [12] P. Gupta, B. Prabhakar, and S. Boyd. Near optimal routing lookups with bounded worst case performance. In *IEEE INFOCOM'00*, 2000.
- [13] H. Hamed and E. Al-Shaer. Dynamic rule ordering optimization for high-speed firewall filtering. In *ASIACCS'06*, 2006.
- [14] H. Hamed, A. El-Atawy, and E. Al-Shaer. Adaptive statistical optimization techniques for firewall packet filtering. In *IEEE INFOCOM'06*, April 2006.
- [15] L. Kencl and C. Schwarzer. Traffic-adaptive packet filtering of denial of service attacks. In *WOWMOM'06: The 2006 International Symposium on World of Wireless, Mobile and Multimedia Networks*, pages 485–489, Washington, DC, USA, 2006. IEEE Computer Society.
- [16] K. Lan and J. Heidemann. On the correlation of internet flow characteristics. Technical Report ISI-TR-574, USC/ISI, 2003.
- [17] L. L. Larmore and D. S. Hirschberg. A fast algorithm for optimal length-limited huffman codes. *J. ACM*, 37(3):464–473, 1990.
- [18] A. J. McAulay and P. Francis. Fast routing table lookup using CAMs. In *IEEE INFOCOM'93*, March 1993.
- [19] Passive Measurement and Analysis Project, National Laboratory for Applied Network Research. Auckland-VIII Traces. <http://pma.nlanr.net/Special/auck8.html>, December 2003.
- [20] V. Srinivasan, Subhash Suri, and George Varghese. Packet classification using tuple space search. In *Computer ACM SIGCOMM Communication Review*, pages 135–146, October 1999.
- [21] D. Taylor and J. Turner. Scalable packet classification using distributed crossproducting of field values. In *IEEE INFOCOM'05*, 2005.
- [22] Thomas Y. C. Woo. A modular approach to packet classification: Algorithms and results. In *IEEE INFOCOM'00*, pages 1213–1222, March 2000.
- [23] A. Wool. A quantitative study of firewall configuration errors. *IEEE Computer*, 37(6):62–67, 2004.