Adaptive Statistical Optimization Techniques for Firewall Packet Filtering

Adel El-Atawy, Hazem Hamed, Ehab Al-Shaer
School of Computer Science, DePaul University, Chicago, USA

***Abstract*— Packet filtering plays a critical role in the performance of many network devices such as firewalls, IPSec gateways, DiffServ and QoS routers. A tremendous amount of research was proposed to optimize packet filters. However, most of the related works use deterministic techniques and do not exploit the traffic characteristics in their optimization schemes. In addition, most packet classifiers give no specific consideration for optimizing packet rejection, which is important for many filtering devices like firewalls.**

Our contribution in this paper is twofold. First, we present a novel algorithm for maximizing early rejection of unwanted flows without impacting other flows significantly. Second, we present a new packet filtering optimization technique that uses adaptive statistical search trees to utilize important traffic characteristics and minimize the average packet matching time. The proposed techniques timely adapt to changes in the traffic conditions by performing simple calculations for optimizing the search data structure. Our techniques are practically attractive because they exhibit simple-to-implement and easy-to-deploy algorithms. Our extensive evaluation study using Internet traces shows that the proposed techniques can significantly minimize the packet filtering time with reasonable memory space requirements.

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 distinct header fields in the packet, which are protocol field, source IP, source port, destination IP , and destination port in IPv4. Each filtering rule R is an array of field values. A packet P is said to match a rule R if each header-field of P matches the corresponding rule-field of R. In firewalls, each rule R is associated with an action 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. A firewall policy consists of N rules $R_1, R_2, \ldots, R_N$. 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 (last rule) is assumed to be deny [1].

With the dramatic advances in the network/link speed, firewall packet filtering must be constantly optimized to cope with the network traffic demands and attacks. This requires reducing the packet matching time needed to "allow" as well as "deny" packets. In fact, discarded packets might cause more harm than others if they are rejected by the default-deny rule as they traverse a long matching path. 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 technologies on the Internet.

In the first part of this paper, we propose a technique that analyzes the firewall policy rules in order to construct a set of rules that can reject the maximum number of unwanted packets (*i.e.*, discarded by the policy rules) as early as possible. This is an NP-complete problem and thus we used an approximation algorithm that pre-processes the firewall policy off-line and generates different near-optimal solutions. However, the most appropriate solution that incurs the least overhead cost is dynamically selected based on the network traffic statistics.

The second part of this work is highly motivated by the Internet traffic properties that were observed in our study in this paper and addressed by other researchers [16] as well. Our study of many Internet and private traces shows that the major portion of the network traffic/flows matches a small subset of the field values in the firewall rules. 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. We therefore propose using statistical search trees based on the matching-frequency of different

field values in the policy, as calculated from the traffic. The ultimate tree is a combination of alphabetic trees optimized on each field-level to consider the frequency distribution of different field values. We presented two tree structures: near-optimal cascaded tree structure for single-threaded processing, and parallel tree structure for network processor platforms. Both the early rejection and the statistical search tree algorithms exhibit lightweight implementations and require no special support in firewalls. They can also be generalized for other filtering devices such as IPSec and Intrusion Detection Systems.

Packet filtering optimization has been studied extensively in the research literature [8]. However, many 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 techniques guarantee an upper bound on the packet matching time. On the other hand, our statistical matching approach aims to improve the average filtering time. In addition, unlike many of the presented techniques, our technique has much less space complexity. The use of statistical trees for optimizing routing table lookups was discussed in [10]. However, the proposed technique uses only a single field (routing prefix) that has a given frequency distribution. Our scheme, however, use statistical tree of multiple-field values that is dynamically updated to reflect the network traffic statistics. Moreover, to the best of our knowledge, the early rejection problem was not addressed by any of the related work.

The paper is organized as follows. In Section II we describe the previously published related work. In Section III we present a technique for early classification of rejected packets. In Section IV we present our statistical-tree based packet classification technique. In Section V we present an evaluation study for the proposed techniques. Finally, in Section VI we present the conclusions and our plans for future work.

II. Related Work

The packet classification problem has been extensively studied recently. The basic approach to packet classification is to sequentially search the rule list until a match is found. Although this approach is very efficient in terms of memory, the scalability of this solution is generally poor, as the search time is proportional to the length of the longest path in the rule list. The main solutions to improve the search times use various combinations of one or more of the following: hardware-based solutions, specialized data structures, geometric algorithms, and heuristics.

Hardware-based solutions using Content Addressable Memories(CAM) exploit the parallelism in the hardware 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 [17], but still limited number of rules. The policy rules are structured as a trie, with a classification time O(B) where B is the total number of bits on all dimensions. This value can still be exceedingly large (*e.g.,* for the 5-tuple in IPv4, $B = 104$).

Aggregated Bit Vector (ABV) approach [2] solves the problem with d independent lookups on one dimension, followed by a combining phase. For each dimension, a lookup is done using a trie, and returning a list of all matching rules on that dimensions. The final result is then computed by finding the rule with highest priority. Because the amount of memory consumed for storing the lists can be extremely large, ABV represents the list using a compressed bit vector.

A wide variety of specialized data structures have been used for fast packet classification. Srinivasan et al. [19] build a table of all possible field value combinations (cross-products) and pre-compute the earliest rule matching each cross-product. Search can be done quickly by doing separate lookups on each field, then combining the results into a cross-product table followed by indexing into the table. Unfortunately, the size of the cross-product table grows significantly with the number of rules.

A geometric algorithm was proposed by Feldmann et al. [6], introducing a data structure called Fat Inverted Segment (FIS) Tree. FIS partitions the first dimension with the endpoints of the projection of the rules on that dimension. Each of the segments is then partitioned, according to the remaining dimensions of the rules covering each segment, into a number of a d dimensional regions. To avoid an O(N^2) explosion of the storage requirements, the d dimensional regions are linked in a Fat Inverted Segment Tree of bounded depth, and the common partitions of the regions 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 based on geometric cutting was introduced by Gupta and McKeown [8] and Woo [21]. Both schemes build a decision tree using local optimization decisions at each node to choose the next bit or field to test. The paper by Woo [21] goes one step ahead by using multiple decision trees. While this may increase search time, it can greatly reduce storage. Similarly, the Hierarchical Cuttings (HiCuts) scheme described in [9] uses range checks instead of bit tests at each node of the decision tree.

Gupta and McKeown [8] 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 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.

The related work closest to our approach is the one proposed by Gupta [10]. By introducing statistical data structures in optimizing packet filtering, this paper became one of the most interesting foundation publications in this domain. In this work, 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 the paper is on routing lookup, the scheme is limited on search trees of a single field with arbitrary statistics. In addition, the paper provides no further details on traffic statistics collection and dynamic update of the statistical tree.

III. Early Traffic Rejection

Firewall rules are often written as exceptions (*i.e.*, accept rules) to the default deny rule for incoming traffic. This might explain the research emphasis on optimizing the acceptance decision path in firewall filtering. However, rejected packets might traverse long decision path of rule matching before they are finally rejected by the default-deny rule. This causes significant matching overhead proportional to the number of rules in the firewall policy. Although packets can be rejected by intermediate deny rules in the policy, we focus on this section on optimizing matching of traffic discarded due to the default rule because it has more profound effect on the performance of the firewall. In addition, optimizing intermediate rule matching is also considered in Section IV. In this section, we will describe a technique that reduces the matching of discarded packets by dynamically introducing an optimal set of early rejection rules in firewall policy. Considering that the amount of rejected traffic is usually less than the accepted traffic, our goal is to select the minimum number of early rejection rules that has the maximum discarding effect (*i.e.*, covering the discard address space in the policy) and they are adaptive to characteristics of the recently discarded traffic. Special consideration was given to allow for fast early rejection with minimum on-line operations.

The address space of the traffic matching the default deny rule (*i.e.*, policy discard space) is obviously the complement of the address space represented by all preceding rules [11]. Intuitively, if a packet does not match any of the field values common to all "allow" rules, then this packet should be rejected as early as possible to save any further matching through the policy. Thus, the early rejection rules (RR) can be formed as a combination of the common field values that cover all rules in the policy. Considering that the number of distinct field values is usually small relative to the policy size (*e.g.*, number of used destination ports is much less than the number of rules), we can show that these rules are more feasible to find. We will search in the firewall policy for a combination of common field values such that every rule uses at least one of these values. For example, if all accept rules use as destination a certain subnet or port number, then packets that do not have this destination address, or destination port can be safely rejected without any further matching.

Let us take S as the set of policy rules, and let S_k^j be the set of all the *rules* having in field f_j the value v_k.

Definition 1: Let $V(f_j)$ = $\{v_k | \exists$ a rule in the policy that have the value v_k for field $f_j\}$, then $S_k^j = \{r_i | f_j^{r_i} = v_k, i = 1 \ldots n, j = 1 \ldots 5\}$ where $f_j^{r_i}$ represents the value of field j in rule i.

Intuitively, S_k^j is the set of rules having in field f_j the same value v_k. The number of different sets (S_k^j) is equal to the number of distinct values ($\sum_{j=1}^{5} |V(f_j)|$) in all the policy rules' fields. The problem is to find a subset of the S_k^j's, such that each rule in the policy is a member of at least one of them.

Definition 2: Let A represents the set of all possible S_k^j, and let $A' \subset A$ represents a selection of S_k^j's such that $\bigcup_{S_k^j \in A'} S_k^j = S$.

This means that the set of rules covered by A' represents all the rules in the policy S.

Theorem 1: The problem of finding the set of field values of minimum size such that each rule in the policy contains at least one of these field values is an NP-Complete problem.

Proof: The decision problem associated with this optimization problem is stated as follows: Finding a set of field value of size at most K such that each rule in the policy contains at least one of these field values.

The proof then follows in the two standard steps:

Step 1: Polynomial Verification of Certificates. Given the solution to the problem we can verify that it correctly covers the whole policy by taking each field value in the

alleged solution and mark down all rules that have this field value. And at the end we perform a quick sweep over the policy to make sure all rules are marked as covered. The complexity of this naive algorithm can be found to be $O(KN)$, where N is the policy size. However, we can have it with less complexity by carefully removing the marked rules after each iteration or sorting the rules by their field values (that will introduce an extra parameter into the complexity expression which is the number of fields specified, which is usually taken as a constant; 4 or 5 is a typical value). The first algorithm will be enough, knowing that K is $O(N)$ we have the algorithm is $O(N^2)$.

Step 2: Reduction from a known NP-Complete problem. We can use the set cover problem (SCP) with bounded element frequency (vertex cover if frequency is only 2). Given a SCP instance: $S = \{e_1, e_2, \ldots e_N\}$, and $S_1, S_2, \ldots S_K \subseteq S$ such that each element e_i can be a member of at most d subsets (the bounded frequency condition). We can solve this instance by an algorithm for our original problem by the following simple mapping: each element e_i is a rule in the policy, that makes S the policy as a whole. The subsets S_j are the subsets of the policy rules having a common field value v_j in one of their fields. By solving for the best (minimum) cover (or a cover of size no more than K) we obtained a solution for the SCP problem.

By Step 1 and 2, our problem of "finding the covering set of field values of minimum size" is an NPC problem. ■

Using a solution A' we can form a Rejection Rule (RR), such that for every $S_k^j \in A'$ there will be a Rejection term (RT) that together compose RR. Hence, we use RR and A' interchangeably throughput this paper.

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

where $Pkt(f_j)$ is the value of field f_j in the packet to be inspected. For example, a typical rule can look like;

$$\begin{aligned} RR \;=\; & (DPort \neq 80) \wedge (DPort \neq 20) \wedge \\ & (DAddr \neq 15.16.17.18) \wedge (Proto \neq UDP) \end{aligned}$$

A. Rejection address-space based optimization

Because it is an NP-Complete problem; searching for the minimum size solution is practically not feasible as the policy size increases. In our specific environment, we limit the size of the set cover to be small (*e.g.*, 1-7 sets) as large set cover solutions will incur unbearable overhead in the filtering of each packet.

We use two approximation algorithms to solve this problem. The first one has an approximation ratio of $1 + \ln(\mid S \mid)$ [14], while the other uses relaxed integer programming and results in an f-approximation ratio [12], where f is the maximum number of subsets that any element can belong to (that in our case will be 5 for the basic form of firewall rules; <proto, srcIP, dstIP, srcPort, dstPort>). The latter algorithm is better for almost all policy sizes (50+ rules) as it gives better approximation ratios, but we use both to help in generating a more diverse set of solutions.

B. Dynamic rule selection

The set cover approximation algorithm generates a set of A_i' to be used as early rejection rules. However, we do not know yet how many and which ones that we should use to achieve an optimal rejection solution in firewall filtering. In this section we will show how we can address both issues using both the policy information and traffic statistics to determine the upper bound and the proper set of A_i''s to be used as RR. It is intuitively clear that the more Rejection rules (equivalently, A_i''s), the more likelihood to reject unwanted traffic as each RR tend to cover more policy discard space. So let us assume for the sake of simplicity that all RR's have the same probability of rejecting a packet, and rejected traffic is only rejected through the default rule (*i.e.*, no traffic is targeted to the deny rules within the policy). Now, take δ_r as the portion of traffic that will be early rejected using r RR's, and $\delta_{\inf}$ as the maximum percentage of the traffic that can be early rejected. Then for the early rejection rules to decrease the average number of comparisons, the number of rejection rules r should be governed by;

$$\begin{aligned} \frac{n}{2}(1-\delta_{\inf}) + n\delta_{\inf} > & \frac{r}{2}\delta_r + (r+\frac{n}{2})(1-\delta_{\inf}) \\ & + (r+n)(\delta_{\inf} - \delta_r) \end{aligned}$$

This leads to:

$$r < \frac{2n\delta_r}{2-\delta_r} \tag{2}$$

The left term in the inequality represents the average number of comparisons per packet without using early rejection, while the first, second and third terms in the right side of the inequality represent average cost of rejection by the early reject rules, acceptance/rejection by the policy and rejection by the default rules respectively.

We can see that the bound on r keeps on increasing for all values of δ, as long as added rules can reject more packets. In the extreme case, where all the traffic is illegitimate, we can have as many RR's as double the policy size ($2n$). This averages to n, which is the number of rules to be checked anyway for each of the packets

Algorithm 1 Startup Phase

$< S, A >\leftarrow$ Convert(Policy Rules)
$r_{max} = \frac{2n\delta_{est}}{2-\delta_{est}}$
$i \leftarrow 0$
repeat
  $A' \leftarrow Approx\_SetCover(S, A)$
  RR_Set $\leftarrow$ Build_Rules(A')
  $i \leftarrow i + 1$
until $i >= r_{max}$ or A' is empty
sort(RR_Set) by size, shorter first
$r \leftarrow 1$
Active_RR_list $\leftarrow$ RR_Set(r)

Algorithm 2 Dynamic Rule Selection

if $\triangle\delta_r > \frac{c(1-\delta_r+\gamma_r)}{2n}$ **then**
  Active_RR_list $\rightarrow$ rule r {Remove last rule, rule r}
  $r \leftarrow r - 1$
end if
if $|RR\_Set| > r$ **then** {More rules to be added}
  $r \leftarrow r + 1$
  Active_RR_list $\leftarrow$ RR_Set(r)
end if
sort Active_RR_list according to hit frequency

Algorithm 3 Early Rejection

Match Packet against Active_RR_list (w' shortcut evaluation)
if packet matched any RR **then**
  reject packet
  INCREMENT $\triangle\delta_i$ of matched rule
  INCREMENT δ_r
else
  send packet to normal filtering process
  INCREMENT γ_r, α_r, or β_r
end if
DECREMENT Window_Expired
if Window_Expired = 0 **then**
  Call Dynamic Rule Selection
  Window_Expired = $Window$
end if

that were going to the default rule if early rejection was not used. So, this implies that we can add extra RR' to optimize filtering as long as the bound is satisfied.

More analysis is needed to determine the effect of adding a specific RR. Let α be the traffic portion accepted by the policy, and after adding r early rejection rules we have δ_r, β_r, and γ_r be the traffic portion rejected by the RR's, the policy *deny* rules, and the default rule respectively. Now, we can state the average number of comparisons/matching per packet after adding the r RR as follows:

$$A_r = c.r(\frac{\delta_r}{2} + \alpha + \beta_r + \gamma_r) + n(\frac{\alpha + \beta_r}{2} + \gamma_r) \quad (3)$$

where c is the relative evaluation cost of an RR which is usually proportional to the number of terms included in the rule. We also have $\partial\delta/\partial r > 0$, $\partial\beta/\partial r, \partial\gamma/\partial r < 0$, and $\alpha + \beta_r + \gamma_r + \delta_r = 1$. Let $\triangle\delta_r$ be the portion of the total traffic that is rejected by the r^{th} RR. Then we can simply show that

$$\beta_{r-1} + \gamma_{r-1} = \beta_r + \gamma_r + \triangle\delta_r \quad (4)$$

To justify adding the r^{th} RR rule: $A_r - A_{r-1} < 0$ must hold. Thus, using (3) and (4) we can derive the following condition:

$$\frac{\triangle\delta_r}{c} > \frac{\frac{\alpha+\beta_r}{2} + \gamma_r}{n} \quad (5)$$

Alternatively, it can be written as

$$\triangle\delta_r > \frac{c\,(1 - \delta_r + \gamma_r)}{2n} \quad (6)$$

to facilitate evaluation at run time according to the type of statistics kept at the firewall. After each window of time, the added rule can be evaluated based on (5) or (6) to decide whether the r^{th} RR is to be used or removed, as described in Algorithm 2.

Initially, we can add the RR's in order of their length, as shorter rejection rule are faster to evaluate and cover more space than longer ones. As the traffic statistics shows the effectiveness of each RR, this will be used periodically to further enhance the operation by dynamically selecting the most valuable early rejection rules. Moreover, RT's are sorted within each rule according to their effectiveness; to optimize running time using shortcut evaluation of each RR's.

The three algorithms show the main operations of the early rejection module. In Algorithm 1, the build up of the candidate rejection rule list out of different solutions to the set cover problem takes place. Algorithm 2 is responsible of the periodic addition/removal of rules according to the performance gain/loss of each rule. Algorithm 3 shows the per-packet operation of filtering; showing the location of early rejection relative to normal packet filtering, as well as the update of statistics required for early rejection. Also, it calls the dynamic rule selection algorithm every window of packets to retune the active early rejection rule list. The use of a pre-processing phase (Algorithm 1) for generating the set cover is to avoid the calculation of new solutions as we go at run time, which can be very computational expensive to be performed real time.

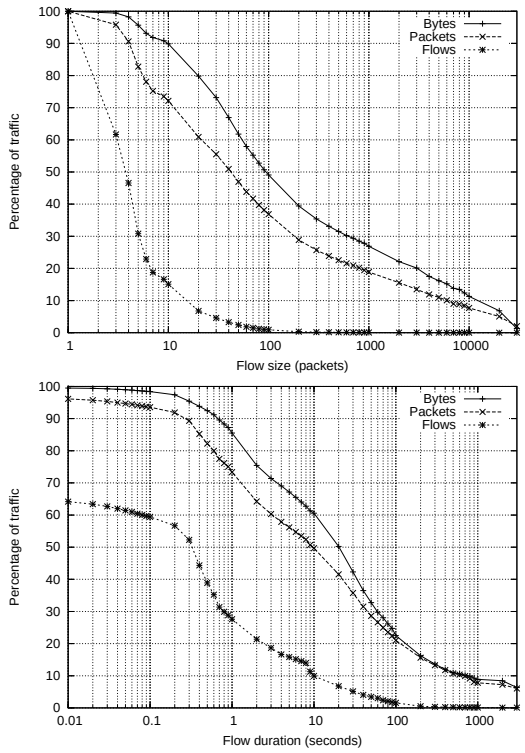

Fig. 1. CCDF distribution of (a) flow size, and (b) flow duration for Internet traces at the University of Auckland.

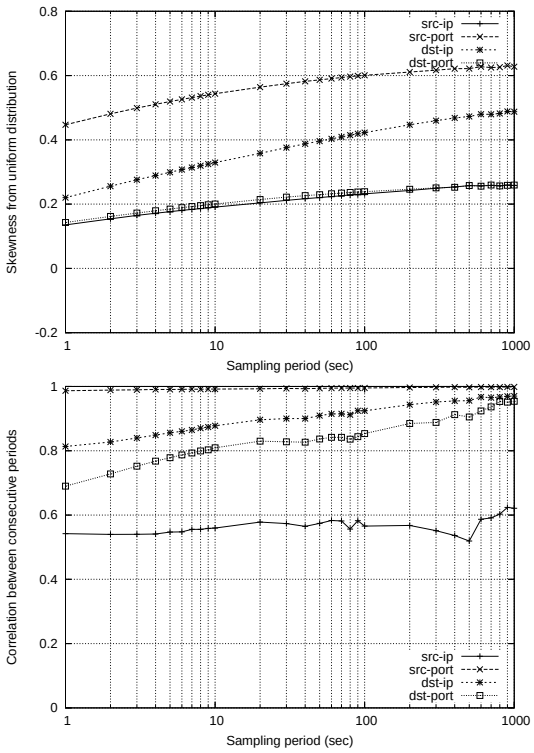

Fig. 2. Analysis of the frequency of packet-header field values: (a) skewness from uniform distribution, and (b) time correlation of the distribution.

IV. Statistical Optimization of Filtering Trees

Although most of the previous work on filtering optimization was based on deterministic techniques, our proposed scheme considers the statistical properties of the traffic passing through the firewall to construct a search tree that gives a near-optimal searching time. We use an adaptive alphabetic tree that dynamically inserts the most frequently used *field values* at the shortest path in the search tree. This results in a significant matching reduction for the most popular traffic. 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, which reveals that the majority of the inbound or outbound packet is matched against a small subset of all filtering field values that exists in the firewall policy. What makes this technique even more attractive is the fact that (1) traffic skewness property is unlikely to change over a short period of time, and (2) the total number of different filtering field values is highly unlikely to be large in a firewall policy, retaining a reasonably shallow alphabetic tree. Thus, a good implementation of this scheme can result in a significant performance gain over the deterministic optimization techniques that use static bounds, as explained in Section II.

Our proposed technique can be summarized as follows. The matching frequency for all filtering rules in the firewall policy is periodically calculated from traffic statistics over intervals of time. These statistics are then used to construct an alphabetic search tree for every filtering field. The constructed trees for each field are combined to obtain an optimal statistical matching tree of all rules in the policy. Finally, the alphabetic tree is updated/reconstructed periodically to match the most-recent traffic characteristics. In this section, we will first present our traffic trace analysis and then describe in detail each one of these steps.

A. Locality of matching properties in firewall filtering

The behavior of Internet traffic retains several characteristics that can be utilized in the optimization of packet filters. In this section, we highlight some observations and properties of Internet flows and packet headers, and we briefly describe how these properties can be useful in reducing the matching time in packet filters. The traffic analysis was performed on several Internet packet traces collected at the edge routers of DePaul University and University of Auckland networks [18]. 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.

1) Packet flow properties: Studying the statistics of various Internet traffic traces, we observed a number of properties for Internet flows.

From our 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% have last 10 seconds or more. It also shows that the long-lived flows carry around 50% of the traffic. Thus, these observations clearly 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. We call this the *locality of flow matching* in firewall filtering. 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.

2) Packet field properties: In this study we analyze the packet-header field values that occur in Internet traffic traces. Studying the statistics of these traces, we observed the following properties for the fields of Internet packet headers.

Skewed field value frequencies: The field value frequency is the number of packets that carry this field value within a certain interval of time. The field frequency distribution is said to be skewed if few field values have high frequencies in comparison to the frequencies of other values in the same time interval. To measure this skewness we use information theory formulation to quantify the Entropy of any given distribution [4]. The skewness factor S_f of a filtering field f is a value between 0 (for a non-skewed or uniform distribution) and 1 (for a totally skewed distribution). S_f is defined by the formula:

$$S_f = 1 - \frac{\sum_{i=1}^{n} p_i \lg p_i}{\lg n} \tag{7}$$

where p_i is the probability of field value v_i and it is calculated as the ratio of the number of packets matching v_i to the total number of packet received. Also n is the number of possible values of field f.

For each traffic trace, the packet-header field frequencies and the skewness of the frequency distribution are calculated for all field values over varying sampling time intervals. We observed that the some fields values are very highly skewed, while other fields have moderate or low skewness. We also observed that the skewness increases slowly when the calculation time interval is increased. Figure 2-(a) shows the skewness of field value frequency distribution of inbound traffic. This figure basically shows that observed values of the source port field have a high skewness factor in the range 0.45-0.6, while the destination address has moderate skewness range of 0.2-0.5, and both the source address and destination port have low skewness of about 0.2. We also call this the *locality of field-value matching* because it shows that only small portion of the field values are used by the majority of the traffic. Thus, it will be highly desirable to place the field values of high skewness/locality as high as possible in the search tree to reduce number of matching for this traffic and eventually the overall packet filtering time.

Time-correlated field value frequencies: To know how long this skewness will last, we study the correlation of the frequency distribution of packet field values over two consecutive time intervals. The field frequency distribution is said to be *time-correlated* if the frequencies of the field value is similar over the two intervals. We use the *correlation factor* C_f of field f as a value between 0 (for an uncorrelated distribution) and 1 (for a totally-correlated distribution), and it is calculated as follows [5]:

$$C_f = \frac{\sum_{i=1}^{n}(p_i - \mu_p)(q_i - \mu_q)}{n \cdot \sigma_p \cdot \sigma_q} \tag{8}$$

where p_i is the probability of field value v_i in a certain time interval, and q_i is the probability in the following interval. The quantities μ_p and μ_q represent the mean, while σ_p and σ_q represent the standard deviation of the probability distributions.

Using the traffic traces, we calculated the correlation of the frequency distribution for varying time intervals. We observed that some packet-header fields retain high time correlation, while other fields have moderate to low correlation. We also observed that the correlation increases slowly with the increase of the time interval for calculating the field frequencies. Figure 2-(b) shows the time-correlation of the field value frequency distribution. This figure shows that source port field has a high correlation factor close to 1.0, while the destination address and port have moderate correlation range of 0.7-0.9, and the source address has low correlation of about 0.6. Therefore, this shows that the field value skewness is

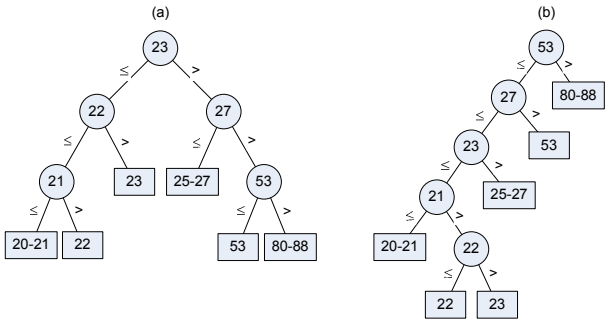

Fig. 3. Search tree for the destination port statistics in Table I: (a) binary tree, and (b) alphabetic tree.

Field	Value	Statistics
dst_port	25-27	0.11
dst_port	23	0.01
dst_port	53	0.19
dst_port	80-88	0.60
dst_port	20-21	0.08
dst_port	22	0.01
...	...	...

TABLE I

EXAMPLE STATISTICS OF THE DESTINATION PORT FIELD.

a valid statistical property that is practically useful to optimize matching against popular filtering field values in the policy for a reasonable period of time.

B. Statistical matching tree

Although binary tree search gives as the worst case search time as $\lg n$ where n is the number of elements, it does not take in consideration the non-uniform (skewed) distribution property of the field values matching based on the traffic characteristics as described in Section IV-A. To exploit this property, a statistical search tree can be built using the values of each filtering field in order to minimize the average matching time. This tree basically inserts values of higher occurrence probability (matching frequency) at higher tree levels than the values with less probability. This way, field values that commonly exist in the traffic will exert less number of packet matches in comparison to uncommon values, resulting in a significant reduction in the matching of most popular flows, reducing the overall average filtering time of all flows.

Although the statistical-based tree matching may not be in favor of less-frequently matched traffic, it still improves the overall average filtering by significantly reducing matching of most popular packets. The more the skewness in the traffic distribution over field values, the more the gain in the filtering performance. Even in the worst case scenario when the traffic distribution is uniform, our techniques cannot do worse than the binary search as a lower bound, which we can argue that is unlikely to occur for all the fields at the same time for long time. Our analysis of large number of various traces presented in Section IV-A supports this. In addition, we will also show how this scheme can be adaptive to track changes in the traffic characteristics over time.

C. Matching tree construction using alphabetic trees

There are several types of statistical search trees that we can employ in our technique. We choose the Alphabetic Search Tree [13] mainly because its construction has low complexity when compared with the Optimal Binary Search tree, and it also has less searching overhead when compared to Huffman trees [15]. Optimal alphabetic binary search trees have been studied for a long time. The best time complexity, for building optimal alphabetic binary trees is achieved by two algorithms: Hu-Tucker [13] and Garsia-Wachs [7]. The resulting tree is an optimal alphabetic binary tree and the complexity of building the tree is O($n \lg n$).

The alphabetic tree stores field values in the leaves based on given weights such that the inherent order of the stored values is preserved. So, at each internal node we can tell that the left subtree contains nodes that have values less than those at the right hand-side. This added constraint of enforcing an order on the placement of values in the tree enables the matching algorithm to branch left or right based on the value extracted from the packet as in the case of binary search trees and eliminates the need for preprocessing of the packet field values.

Figure 3-(a) shows the normal balanced binary search tree for the destination port filtering field values given in Table I. The corresponding statistically optimized tree is shown in Figure 3-(b). Using the field statistics given in the table, the average number of matches is 3.8. The average matching is reduced to 2.8, which is above 26% reduction from the binary tree case. Notice that every node in the binary search tree contains non-overlapping values or range of values. The firewall policy can be easily pre-processed to resolve any conflict between overlapping values of the same filtering field [3].

Alphabetic tree aggregation: Since packet matching is performed on multiple fields (5-tuple), multiple alphabetic search trees are constructed to correspond to source IP, source port, destination IP and destination port. The four trees are combined together to form a statistical matching tree implementation based on alphabetic trees. We propose two approaches to achieve this goal: cascaded-search and parallel-search trees.

In the cascaded search approach (Figure 4-(a)), we start by building the top-level alphabetic tree using the

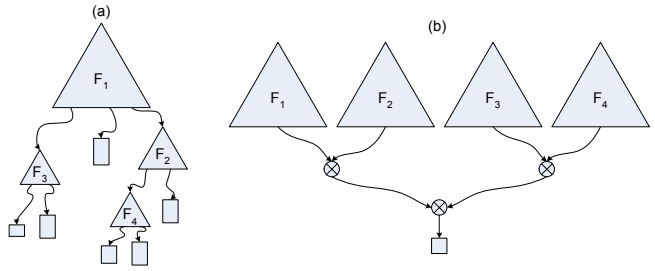

Fig. 4. Aggregate matching tree structure for (a) cascaded matching, (b) parallel matching.

Algorithm 4 BuildTree (S, F)

S: Set of rules
F: Set of filtering fields
if $|S| < limit$ **then**
  return S
end if
for all $f_j \in F$ **do**
  for all $v_k \in V(f_j)$ **do**
    $C(v_k) = \sum_{r_i \in S_k^j} C(r_i)$
  end for
  Calculate S_{f_j}
end for
Choose $f_{best} : S_{f_{best}} \leq S_{f_j} \forall f_j \in F$
T = Construct Alphabetic Tree for f_{best}
for all $v_k \in V(f_{best})$ **do**
  T' = $BuildTree(S_k^{f_best}, F - \{f_{best}\})$
  $T.v_k \leftarrow T'$ {Saving a pointer to sub-tree at leaf node}
end for
return T

filtering field of highest skewness. The field skewness is calculated based on (7) using the number of packets matching each field value during a specific time interval. For each field value v_k, the packet count is collected by summing the number of packets hitting all the rules that carry this value. This information is normally recorded by filtering devices and are readily available at no extra processing cost. Each leaf in the top-level tree holds the field value, as well as a pointer to another alphabetic subtree built recursively using values of the field that has next highest skewness considering only the rules that use this field value. As shown in Algorithm 4, the cascaded-search tree construction continues in all leaves/values until trees for each field are constructed (*i.e.*, 4 levels of cascaded trees if we exclude the protocol field) to represent the entire set of rules in the firewall policy. It is important though to notice that it may not be necessary to build a tree in each level particulary if the number of field values remaining is too small to gain a benefit over the linear or binary search.

Theorem 2: Given a policy of n rules, and a constant number of filtering fields. The construction of the cascaded matching tree structure using Algorithm 4 has a time complexity of $O(n.\lg(n))$.

Theorem 3: Given a policy of n rules, and a constant number of filtering fields. The cascaded matching tree structure has a space complexity of $O(n)$.

Proof: The intuition behind both theorems is that the sum of the number of leaves of the trees at any level l is bounded by the number of rules. This is attributed to the fact that each leaf represents a unique combination of field values from the top level down to level l, and the number of such combinations can not exceed the total number of rules in the policy. Thus, the total size of internal and external nodes in all the trees at any given level cannot exceed $2n$. Also, the time complexity of building all the trees at a certain level cannot exceed the cost of building a single tree with the size of all trees combined ($O(n.\lg(n))$). The total space/time complexity is the complexity of a single level multiplied by the number of levels (fields). Since the number of filtering fields is a constant, the space complexity is $O(n)$, and the time complexity is $O(n.\lg(n))$.

To prove the above intuition formally: For any given field f_j, the set of field values is bounded by the number of rules (*i.e.*, $|V(f_j)| \leq n$). By building the cascaded tree structure, we will have a set of trees each contributing with a certain number of leaves. At level l, we have L_l leaves, each leaf might be followed by one more tree, or a simple short list of rules. At the first level, we have a single tree based on a chosen field (with minimum entropy). The number of leaves of this initial tree is bounded by the maximum size of field values (*i.e.*, $L_1 \leq n$). Each field value will receive a subset of the rules to further process. So the maximum number of leaves in each of the trees of this next level is bounded by the size of its rule subset: $n_1, n_2, \ldots n_m$. Moreover, these subsets' sizes sum up to a value less than n ($L_l = \sum n_i \leq n$). It is a known fact that $\sum g(n_i) < g(n) if g(.) = \omega(x)$. Therefore, we deduce that the building time of all the trees in the second level is $O(n \lg(n))$. Having this fact valid for all successive trees (to a maximum of 5 levels corresponding to the number of fields), we conclude that the overall building time is $O(dn \lg(n)) = O(n \lg(n))$ for a constant d. For the space complexity we follow a similar argument, but regarding the size of each of the trees on any level. We know that the number of nodes in any n_i leaves tree will be $2n_i - 1$ nodes (internal and external). Thus the overall space complexity will be $O(dn) = O(n)$ for a constant number of fields. ■

As an alterative approach, we also developed a parallel tree structure (Figure 4-(b)) that constructs an alphabetic tree for every filtering field. All the tress, four in our case,

Algorithm 5 Cascaded-tree filtering

```
T_i ← Top-level search tree
ref ← Lookup H in tree T_i
if ref is a tree for field f_j then
    T_i ← ref
    f_i ← f_j
    Goto 2
else if ref is a list L then
    rule ← Lookup H in list L
    if rule ≠ nil then
        action ← rule[action]
    else
        action ← DEFAULT
    end if
end if
```

Algorithm 6 Parallel-tree filtering

```
for each field f_i in set of optimized fields do
    rules[f_i] ← Lookup H[f_i] in tree T_{f_i}
end for
candidates ← φ
for each list L_j in rules do
    candidates ← candidates ∩ L_j
    if candidates = φ then
        action ← DEFAULT
    end if
end for
rule ← Lookup H in candidates
if rule ≠ nil then
    action ← rule[action]
else
    action ← DEFAULT
end if
```

can be searched in parallel and the matching results are then combined to produce the final matching results, as discussed in Section IV-D. The parallel tree structure has the advantage of executing multiple searches concurrently particularly on a network processor or multi-threaded hardware. However, the cascaded tree structure always gives less number of matches particulary if the skewness factor varies significantly between of the different fields. We discuss this issue in more detail in our evaluation experiments in Section V.

D. Policy matching algorithms using alphabetic tree

After constructing the search trees, we proceed with the packet matching process. The matching operation is performed for each packet header field against the list of field values in the filtering rules. In order to perform the search on multiple fields (5-tuples), we have two approaches depending on the underlying tree search structure as follows.

Cascaded-tree matching: In this approach, we use the cascaded search tree described in Section IV-C. The algorithm starts with looking up the packet header value in the top-level search tree of the highest skewness field. As a result, the matching leaf node returns a reference to either a search subtree for another field or a list of rules that carry the field value in this leaf. In the former case, the referenced tree is searched recursively for a matching field value. In the latter, the list is linearly searched for a matching rule. Once a rule matches the packet, the corresponding action is returned, otherwise the search continues till the end of the list. If no matching rules are found, the default filtering action is returned.

Although the algorithm involves linearly searching a list of rules at the final lookup stage, the matching operations are very limited because the list contains only a small number of rules as discussed in Section IV-C.

Parallel-tree matching: In this approach, the parallel search tree described in Section IV-C is used. Packet lookup is performed against each of the field search trees separately. As a result, we obtain for each field a set of candidate rules that contain the corresponding field value. Then, the rule that matches the packet is found by getting the intersection between these sets of rules. If the intersection contains more than one rule, the rule with highest order (priority) is selected. If no rules are common, then the default action is returned.

E. Tree reconstruction and updates

After the alphabetic trees are constructed for eligible fields, they are used for matching upcoming packets. The reduction in matching is maximal when the upcoming traffic distribution over field values exactly matches the distribution when the tree has been constructed. However, this is not very likely to happen since as time passes, some flows start and others terminate, leading to accumulative changes in the traffic distribution over field values. Therefore, using an alphabetic tree with very old field value probabilities may result in inefficient matching that yields more average search time than regular binary search. To avoid this situation, we impose two types of rectification to the alphabetic tree; performance triggered updates, and periodic mandatory updates. The first update is performed more frequently and basically rebuilds the tree when traffic dynamics lowers the performance (increases average number of comparisons) below (above) a certain threshold. The second rectification is executed at larger intervals and simply disposes outdated alphabetic trees, and constructs more effective ones based on current statistics.

Performance triggered updates: An accurate measure for the effectiveness of using the alphabetic tree to search the values of a certain field f is the *optimization efficacy* ε_f. The quantity ε_f is defined as the actual reduction in matching as compared to binary search when the current traffic is matched against an alphabetic search tree built using the traffic statistics collected in the previous time interval. Mathematically, ε_f is given by the following formula:

$$\varepsilon_f = 1 - \frac{\sum_{i=1}^{n} q_i \lg p_i}{\lg n} \tag{9}$$

where q_i is the probability of field value v_i in the current time interval, and p_i is the probability of this value in the preceding interval.

Although this formula accurately estimates the matching gain, its calculation is very expensive to be performed at runtime for every packet. Another lightweight measure that gives very close average results is the exponential moving average of the matching gain $\overline{\varepsilon}$.

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

$$\overline{\varepsilon}_i = (1-\omega)\overline{\varepsilon}_{i-1} + \left(\frac{\omega}{\lg n}\right) h_i - \omega \tag{10}$$

$$\overline{\varepsilon}_{thr} = \tau \varepsilon_{opt} \tag{11}$$

where h_i is the height (*i.e.*, number of comparisons) of the destination leaf of packet i, g_i is the gain over binary search for packet i. After a packet is matched using the alphabetic tree, if $\overline{\varepsilon}_i$ drops below a certain threshold $\overline{\varepsilon}_{thr}$, the alphabet search tree is disposed for this field and a new tree is built. $\overline{\varepsilon}_{thr}$ is calculated as a ratio τ of the optimal gain ε_{opt} that the tree was built to achieve. Notice that these expressions involve only inexpensive addition and multiplication operations. Figure 11 in section V-B shows these updates and their effect on the performance of packet matching.

Periodic mandatory updates: To avoid extended 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 field value statistics in order to boost back the matching performance close to its optimum level. Since this type of tree update is mandatory, the update period should be determined based on the computational capacity of the filtering device. A reasonable update period that suits common firewall devices could be one hour.

Policy Size (approx)	Acceptance Rate (α)	Average $\triangle\delta_r$	Opt. number of RR's	Gain
500	75%	3%	30	7.3%
500	50%	3%	30	24.7%
500	50%	7%	36	34.9%
1000	85%	5%	33	9.20%
1000	75%	5%	44	18.3%
1000	50%	5%	58	37.8%
230	25%	5%	33	32.1%
230	50%	5%	27	18.8%
230	75%	5%	12	3.60%
570	25%	7%	41	47.8%
570	50%	7%	47	33.2%
570	75%	7%	27	15.2%
570	85%	7%	21	7.1%

TABLE II
EFFECT OF EARLY REJECTION ON AVERAGE NUMBER OF COMPARISONS

Policy	Number of Rules	Accepted Traffic (α)
Policy 1	1000	50%
Policy 2	1000	75%
Policy 3	200	50%

TABLE III
TEST POLICIES USED FOR GENERATING THE PLOTS IN FIGURE 5

V. PERFORMANCE EVALUATION

A. Evaluation of early rejection

In this section, we evaluate the performance of the early rejection technique. To test the performance gain of using early rejection, we used several real and generated policies with varying traffic behavior. The traffic was injected to the policy, and the number of matches was calculated.

Table II shows the gain of using early rejection rules on the average number of matches (*i.e.*, against policy rules or rejection rules) using different policies; real and generated (In Section V-B, we describe our technique of generating policies). We injected tailored traffic that has different percentages of accepted versus rejected flows, and also varying matching probability with the rejection rules to study its effect. The gain was high enough in many cases which manifests a noticeable performance increase in the firewall operation. The results were very encouraging in all test cases, but when very small-sized policies were used, the results were varying widely depending on the current pattern of traffic used.

In the next experiment, we studied the effect of how much of the traffic is rejected by the default rule

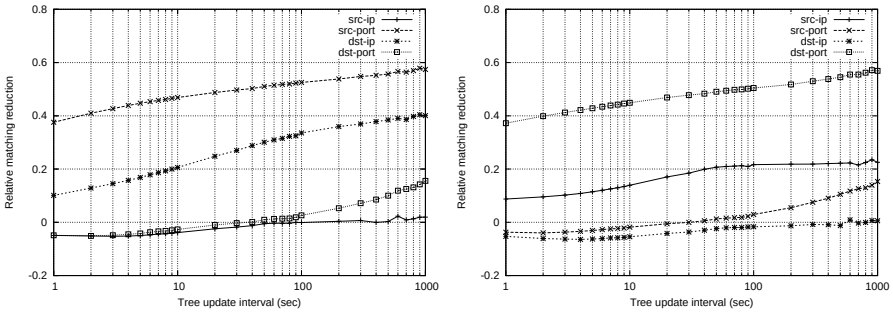

Fig. 6. The reduction of packet matching relative to binary search for each filtering field on the firewall (a) inbound interface, (b) outbound interface.

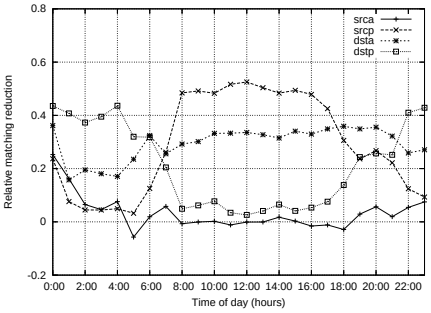

Fig. 7. Relative matching reduction for each field for different times of day

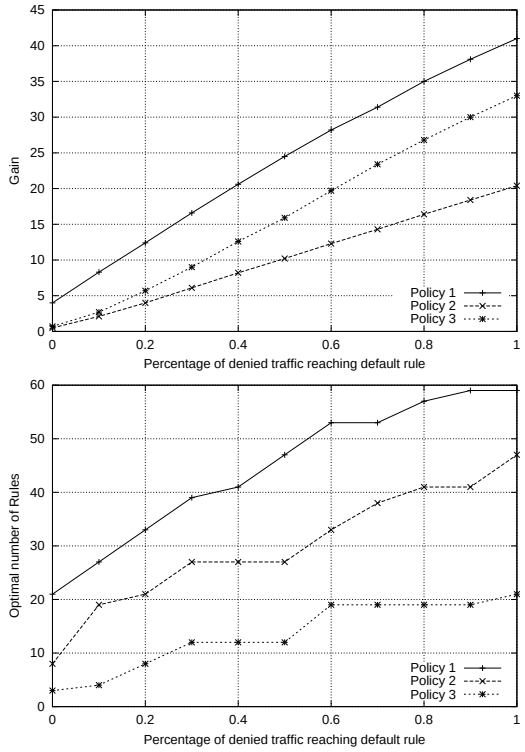

Fig. 5. Early rejection (a) performance gain, and (b) the number of Rejection Rules for three policies with varying percentage of default rule traffic.

versus policy deny rules. Intuitively, the more the traffic reaching the default rule the more successful the early rejection technique will be. This was verified clearly by injecting traffic tailored with different portions targeted to the default rule. Figures 5 show the gain as it increases when the portion that can reach the default rule increases from 0% to 100%. As RR's are added, the traffic reaching the default rule decreases which increases the gain of adding such early rules. These results were obtained using three policies and associated traffic as shown in Table III. Taking into consideration the percentage of the accepted traffic, we can see that the achieved gain in performance was not far from the optimal gain. For example, in *Policy 1*, the gain has reached 41% where the optimum is 50% which can only be achieved if it was possible to early reject all the traffic with no overhead of early rejection rules.

B. Evaluation of adaptive statistical filtering

In this section, we evaluate the performance of the alphabetic tree filtering technique. We use real-life packet traces obtained from the NLANR project [18]. Based on the traffic flow information, we generated filtering rules to handle inbound and outbound traffic to the network. For each filtering field, we use 256 different values extracted from the packet header information in the trace. Different filtering rules are composed by randomly mixing and matching different field values in order to generate a total of 2000 rules. The generated policy typically resembles medium sized firewall rules existing in real firewall policies [22]. These filtering rules are used to study the effectiveness of our proposed technique. We measure the packet matching performance by evaluating the reduction in the number of field matches relative to balanced binary search tree instead of using the absolute number of matches. Since binary search trees require a constant number of matches (8 in our policy) to lookup any field value, the relative reduction in matching is directly proportional to the actual number of matches. However, relative match reduction has the advantage of being normalized, making it more suitable for observing and studying the performance of our technique.

1) Optimization effectiveness for individual filtering fields: In this experiment, for each individual filtering field, we evaluated the average relative match reduction when using our technique during various tree update intervals. The traffic used in the experiment is collected from the inbound firewall interface on a weekday between 12:00pm and 12:59pm. The results are presented in Figure 6.

We observe that our technique outperforms the binary search for certain field types, but very close to binary search for other fields. For inbound traffic in Figure 6-(a), applying our technique to the source port field resulted in 40% to 60% gain over binary search, while for the destination address the gain varied from 10% to 40% depending on the length of the tree update interval. On the other hand, the destination port and source address performance was almost similar to the binary search. Similarly, for outbound traffic in Figure 6-(b), the highest performance gain was (40%-60%) for the destination port field, and (10%-20%) for the source address. However, the source port and destination address did not show any significant gain.

Another observation is that increasing the tree update interval improves, but slowly, the average match reduction. However, an over-extended update interval does not significantly improve the matching gain. This is observed in the same figure, where the source port matching gain increases from 40% to 54% when the update interval is increased from 1 to 100 seconds, however, increasing the period 10 times results in only 6% increase in matching gain. These results can be attributed to the fact that a longer update interval gives more accurate statistics of field values, while extending that interval adds only a little more information. The collected statistics is closely coupled with the Internet flow dynamics characterized by flow lifetimes of less than 10 seconds as shown in Figure 1.

In another experiment, we recorded the relative matching reduction for different filtering field types during a full weekday interval between 12:00pm and 12:59pm. The results for the experiment are shown in Figure 7. As previously noted, we observed that at most two filtering fields sustain a high degree of skewness simultaneously, which improves the filtering efficiency throughout most of the day. Another important observation is that the highly search effective fields change with time of day. For example, during rush hours (8:00am-4:00pm), the source port and the destination address are the most effective fields, while at late night (10:00pm-4:00am) the destination port is more effective. This emphasizes the importance of dynamically choosing the most effective (skewed) filtering field on tree updates.

2) Optimization effectiveness for filtering policy: In this experiment, we evaluate the overall average relative reduction in packet matching for the inbound filtering policy using our technique with varying tree update interval. The traffic used in the experiment is collected on a weekday between 12:00pm and 12:59pm. The experiment is performed for the both the cascaded and parallel search implementations.

Figures 8-(a) and 8-(b) show that, using our technique, the average relative match reduction is very close to the optimal case, when the field value statistics in a given interval exactly matches the statistics used in the previous interval to build the alphabetic tree. The deviation from the optimal case is due to the fact that the field value statistics are constantly changing with the Internet traffic dynamics. We also observe that, similar to individual fields, the average overall relative match reduction increases logarithmically with the tree update interval, while the variance decreases. The highest observed average relative match reduction measurements are 0.5 and 0.4 with a 400s update interval for cascaded and parallel search respectively. Beyond this update interval, the match reduction average and variance are almost constant.

To study how frequent our technique achieves different performance gain levels, we provide a cumulative statistics of the number of measurements versus the relative match reduction in Figures 9-(a) and 9-(b). For example, the plots show that, for 100s update interval, 90% of the measurements reflect better than 0.4 and 0.3 relative matching gain in the cascaded and parallel search respectively.

Figures 10-(a) and 10-(b) show the performance of our techniques with different tree update intervals for an extended period of time from 12:00am to 11:59pm on a weekday. It is clear from this figure that the relative matching gain does not persist at a specific level for a long period of time. The maximum gain is achieved during day hours where fast filtering is highly needed, due to the existence of a large traffic volume that consequently creates significant skewness in the field value statistics. During evening and night hours, the traffic flow is much less, hence a reduced degree of skewness as well as in the relative match reduction. The observations regarding the tree update interval are consistent throughout the entire day period with our observations for the one hour interval.

In all these experiments, the parallel tree search consistently resulted in less match reduction than the cascaded search. This can be explained in the context of data structure sizes used in both cases. In the cascaded search, the top-level tree provides the smallest number of matches, while the lower-level cascaded trees only add a relatively small number of matches to that. In the parallel search, the effective performance is determined by the largest number of matches needed by any of the search trees. In our experiment, this number turns out to be significantly larger than the smallest number of matches. This is because there is a large difference between the skewness of the mostly skewed field and the next one.

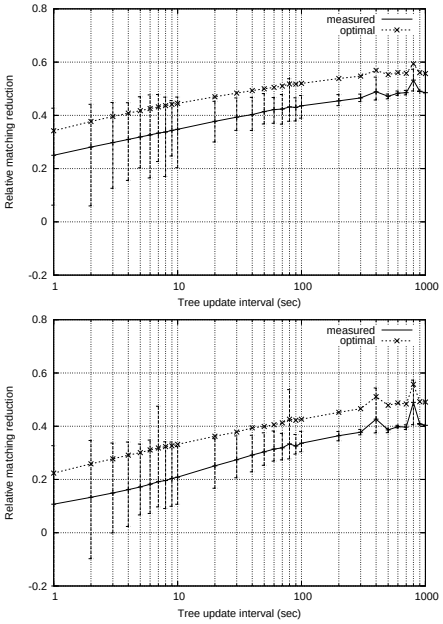

Fig. 8. Average optimal and measured relative matching reduction with varying update interval for (a) cascaded search, (b) parallel search.

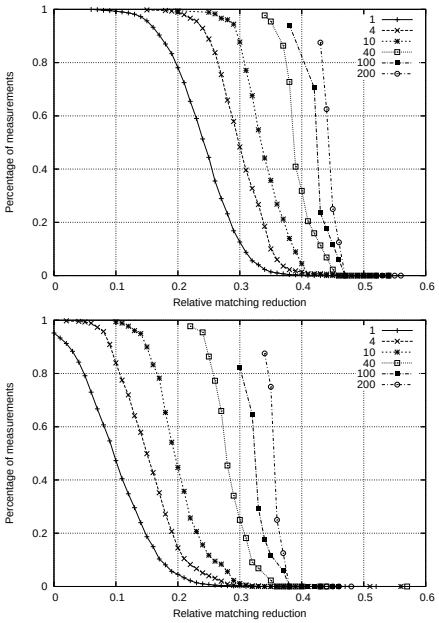

Fig. 9. Cumulative ratio of measurements greater than different matching reduction for (a) cascaded search, (b) parallel search.

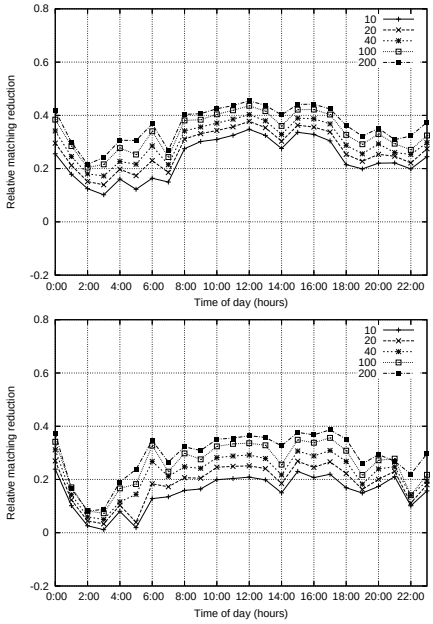

Fig. 10. Measured matching reduction for a full day interval with different update intervals for (a) cascaded search, (b) parallel search .

Therefore, if the skewness is very similar for all fields, the parallel search would have surely outperformed the cascaded search.

3) Adaptive tree updates: In this experiment, we closely examine the dynamics of the adaptive search tree update mechanism during a one hour interval on a weekday from 12:00pm to 12:59pm. The field value statistics are collected in the first 20 seconds, then the matching efficacy of the most skewed field (source port) is calculated periodically every 20 seconds based on the formulas in Section IV-E. For each interval, the search tree efficacy ε is evaluated, and the weighted moving average of the tree efficacy $\bar{\varepsilon}$ is updated. The alphabet tree is dynamically reconstructed whenever the average tree effectiveness deviates significantly from the optimal depth ($\tau = 0.2$). Figure 11 shows the results of this experiment with tree updates indicated by the solid triangular marks.

The graph shows the average tree efficacy is updated smoothly at every tree update interval, thus ignoring sudden short-term decrease in the instantaneous matching efficacy. When the matching efficacy trend sustains a continuous decline, the average efficacy falls below the designated threshold and the tree is reconstructed based on the most recent statistics. Tree updates are computationally intensive and should be performed only when crucially needed. The frequency of tree updates are tightly coupled with the deviation threshold τ and the averaging smoothing factor ω. Our study of various settings of these parameters show that, during rush hours, our adaptive technique reconstructs the tree only 2-5 times in an hour when $\omega = 0.2$ and $\tau = 0.2$. This incurs minor amortized overhead throughout the full interval in which the alphabet tree is utilized.

VI. Conclusion

The Packet classification optimization problem has received the attention of the research community for many years. Nevertheless, there is a manifested need for new innovative directions to enable filtering devices such as firewalls to keep up with high-speed networking demands. This paper addresses two important problems related to packet filtering that are not yet thoroughly explored in research: (1) early rejection of unwanted packets, and (2) optimizing packet filtering based on traffic statistics. The paper presents techniques, algorithms and evaluation study to tackle each problem effectively.

As the size of a firewall policy grows, the effect of discarded packets by default-deny rule become increasingly harmful. we propose a novel technique that introduce a minimal overhead on the firewall processing to allow rejecting the maximum number of these packets as early as possible, thereby reducing the matching time significantly. We use an approximation algorithm off-line to generate a set of near-optimal solutions based on the firewall policy. Each one of these solutions represents an early rejection rule to be inserted at the beginning

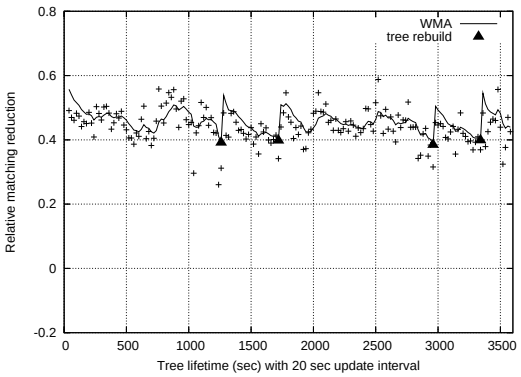

Fig. 11. Relative matching reduction the source port during one hour interval

of the policy to discard traffic intended for the default-deny rule. Then, we use a dynamic algorithm to select the early reject rule set that results in a minimum filtering cost based on the current traffic statistics. Our early rejection technique shows a matching reduction of 19% when the discarded traffic is as low as 25% of the total traffic, and 50% when the total discarded traffic is as high as 75%. The number of added early rejection rules is relatively small: $4\% - 10\%$ of the size of the original firewall policy.

For the second problem, we first show that the matching-frequency of field values in firewall rules is a profound property to utilize in statistical packet matching. We use this property to construct an aggregate alphabetic trees that is adaptive to changes in the traffic characteristics while considering the tree maintenance cost. We also show that for the aggregate cascade tree structure the space complexity is bounded by $O(n)$, and computational complexity is $O(n \lg n)$. We can argue that using statistical alphabetic filtering tree guarantees obtaining near-optimal average matching time if the traffic statistics get stable over time. Therefore, this paper addresses key issues related to this problem including identifying practically useful statistical properties, building an optimal statistical filtering tree over multiple fields for single and multi-threaded matching implementations, and dynamic tree update based on recent statistics. Our evaluation of the statistical filtering tree approach using the traffic traces shows that, during day hours, with 200 sec update interval, the cascaded tree achieves 45% average relative gain, while the parallel tree achieves 35%. However, during the evening hours, the cascaded tree achieves only $20\%-30\%$. On the other hand, the tree reconstruction keep the relative gain close to optimal, while being infrequent (about 2-5 times/hour).

The implementations of both techniques are simple and lightweight. The statistics collection is simple calculation (e.g., counter increments) based on easily obtainable information from well-known utilities like Netflow [20].

REFERENCES

[1] E. Al-Shaer and H. Hamed. Discovery of policy anomalies in distributed firewalls. In *IEEE INFOCOM'04*, March 2004.

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

[3] F. Baboescu and G. Varghese. Fast and scalable conflict detection for packet classifiers. *Computer Networks*, 42(6), 2003.

[4] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & sons, 1991.

[5] A. L. Edwards. *An Introduction to Linear Regression and Correlation*. W. H. Freeman and Co, San Francisco, 1993.

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

[7] A. Garsia and M. Wachs. A new algorithm for minimum cost binary trees. *SIAM Journal on Computing*, 6(4):622–642, 1977.

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

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

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

[11] H. Hamed, E. Al-Shaer, and W. Marrero. Modeling and verification of IPSec and VPN security policies. In *IEEE ICNP'05*, Nov. 2005.

[12] D.S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, pages 555–556, 1982.

[13] T. C. Hu and A. C. Tucker. Optimal computer search trees and variable length alphabetic codes. *SIAM Journal on Applied Mathematics*, 21:514–532, 1971.

[14] D. S. Johnson. Approximation algorithms for combinatorial problems. In *STOC '73: Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 38–49, 1973.

[15] D. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, Massachusetts, second edition.

[16] K. Lan and J. Heidemann. On the correlation of internet flow characteristics. Technical Report ISI-TR-574, USC/ISI, 2003.

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

[18] Passive Measurement and Analysis Project, National Laboratory for Applied Network Research. Auckland-VIII Traces. http://pma.nlanr.net/Special/auck8.html, December 2003.

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

[20] Cisco Systems. Netflow services solutions guide. http://www.cisco.com, Oct. 2004.

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

[22] A. Wool. A quantitative study of firewall configuration errors. *IEEE Computer*, 37(6):62–67, 2004.