

Dynamic Rule-ordering Optimization for High-speed Firewall Filtering

Hazem Hamed and Ehab Al-Shaer

School of Computer Science, Telecommunications and Information Systems
DePaul University
Chicago, Illinois, USA

{hhamed, ehab}@cs.depaul.edu

ABSTRACT

Packet filtering plays a critical role in many of the current high speed network technologies such as firewalls and IPSec devices. The optimization of firewall policies is critically important to provide high performance packet filtering particularly for high speed network security. Current packet filtering techniques exploit the characteristics of the filtering policies, but they do not consider the traffic behavior in optimizing their search data structures. This results in impractically high space complexity, which undermines the performance gain offered by these techniques. Also, these techniques offer upper bounds for the worst case search times; nevertheless, average case scenarios are not necessarily optimized. Moreover, the types of packet filtering fields used in most of these techniques are limited to IP header fields and cannot be generalized to cover transport and application layer filtering.

In this paper, we present a novel technique that utilizes Internet traffic characteristics to optimize firewall filtering policies. The proposed technique timely adapts to the traffic conditions using actively calculated statistics to dynamically optimize the ordering of packet filtering rules. The rule importance in traffic matching as well as its dependency on other rules are both considered in our optimization algorithm. Through extensive evaluation experiments using simulated and real Internet traffic traces, the proposed mechanism is shown to be efficient and easy to deploy in practical firewall implementations.

1. INTRODUCTION

Firewalls and IPSec gateways have become major components in the current high speed Internet infrastructure to filter out undesired traffic and protect the integrity and confidentiality of critical traffic. In these devices, the filtering decision is based on a *security policy* defined according to predefined high level security requirements and is composed of a set of ordered filtering rules against which the net-

work traffic is sequentially matched in order to determine the appropriate filtering action. Therefore, packet filtering is a critical component that determines the performance of these networking devices. With the dramatic advances in the current network speeds, 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” or “deny” packets in order to minimize the end-to-end delay. This problem is even more critical for application-level filtering, where a wide variety of packet filtering fields is used. Thus, efficient yet easy to implement filtering policy optimization is highly crucial to enable high speed packet filtering for effective deployment of traffic filtering technologies in the Internet.

Our work in this paper is highly motivated by a number of Internet traffic properties that we observed in our study and addressed by other researchers [4, 12] as well. Our study of many Internet and private traces shows that the major portion of the network traffic matches a small subset of the firewall rules. We also observed that this “skewness” in traffic distribution over policy rules is likely to stay for time intervals that are sufficient to make such skewness important to consider in packet filtering. Thus, in this paper, we show that the rule ordering optimization problem with rule dependency constraints is NP-complete, and we present a novel *dynamic rule ordering technique* that employs the matching skewness of firewall rules to enhance filtering performance. Our ordering scheme splits the filtering policy into two layers of rules. The top layer consists of a small set of the most *active rules* (*i.e.*, currently performs the most packet matching) ordered based on their traffic matching ratio to minimize the overall packet matching. The second layer usually contains a larger set of the remaining *inactive rules* that perform much less matching. The optimized rule ordering is constantly being updated in order to adapt to the new traffic statistics. Our technique exhibits lightweight implementation and can also be generalized for other filtering devices such as IPSec and Intrusion Detection/Prevention Systems.

Packet filtering policy optimization has been studied extensively in the research literature [7]. 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 match-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ASIACCS'06, March 21-24, 2006, Taipei, Taiwan.
Copyright 2006 ACM 1-59593-272-0/06/000 ... \$5.00.

| protocol | source address:port | destination address:port | action |
|----------|---------------------|--------------------------|--------|
| 1 tcp, | *.*.*.*:any, | 161.120.33.41:25, | allow |
| 2 tcp, | 140.192.37.30:any, | *.*.*.*:21, | deny |
| 3 tcp, | *.*.*.*:any, | 161.120.33.*:21, | deny |
| 4 tcp, | 140.192.37.*:any, | *.*.*.*:21, | allow |
| 5 tcp, | *.*.*.*:any, | 161.120.33.*:22, | allow |
| 6 tcp, | 140.192.37.*:any, | *.*.*.*:80, | deny |
| 7 tcp, | *.*.*.*:any, | 161.120.33.40:80, | allow |
| 8 tcp, | *.*.*.*:any, | 161.120.33.43:53, | allow |
| 9 udp, | *.*.*.*:any, | 161.120.33.43:53, | allow |

Figure 1: A typical firewall packet filtering policy example.

ing approach aims to improve the average filtering time. In addition, unlike many of the presented techniques, our technique has much less space complexity and can be used with any filtering rule formats including application-level filtering policies. The use of statistical structures for optimizing routing table lookups was discussed in [9]. However, the proposed technique uses only a single field (routing prefix) that has a given frequency distribution. Optimization of decision lists has been briefly discussed in [4]. The work was mainly focused on optimizing the space complexity of packet filtering decision trees, and used a simple greedy algorithm for ordering filtering rules as one step in the approach. However, the greedy algorithm only works only when most of the filtering rules are disjoint. None of these techniques explain how these optimizations can be implemented. Our scheme, however, uses statistically optimized ordered rule lists of multiple-field types that is dynamically updated to reflect the network traffic statistics.

The paper is organized as follows. Section 2 presents the background and definitions related to firewall packet filtering. In Section 3 we present a study of Internet flow characteristics as a motivation for our work. Section 4 describes the problem of optimizing firewall rule ordering and our proposed heuristic optimization algorithm. In Section 5 we present the implementation details of our firewall dynamic rule ordering mechanism. In Section 6 we present the results of our evaluation experiments for the proposed mechanism. Section 7 studies related research in the area. Finally, in Section 8 we conclude our work.

2. FIREWALL PACKET FILTERING BACKGROUND

The main task of packet filters in security policies is to categorize packets based on a set of rules representing the filtering policy. The information used for filtering packets is usually contained in distinct header fields in the packet, which are the protocol, source IP, source port, destination IP, and destination port IPv4 fields. 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 $R=(\text{TCP}, 140.192.*:23, *.*.*, \text{allow})$ matches traffic destined to subnet 140.192 and TCP destination port 23 only.

A firewall policy typically consists of an ordered list of n packet filtering rules R_1, R_2, \dots, R_n such that packets are sequentially matched against these rules until a matching rule is found. 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 [3]. The filtering rules may not be disjoint, and thus packets may match one or more rules in the firewall policy [1]. In this case, these rules are said to be *dependent* and their relative ordering must be preserved for the firewall policy to operate correctly. For any two dependent rules that have different filtering actions, the rule(s) that must have smaller order is said to have higher *precedence* relative to the following related rule(s). The example filtering policy shown in Figure 1 includes a dependency between rules 2, 3 and 4, and between rules 6 and 7. The *dependency ratio* of a filtering policy is defined as the ratio of rules that depend on subsequent rule(s) other than the default, and the *dependency depth* is the average number of rules involved in the dependency relations. In our example policy, the dependency ratio is $\frac{2+1}{9}$ and the dependency depth is $\frac{3+2}{2}$.

Typically, a firewall matches all incoming and outgoing packets against the rules in the firewall filter table in sequential order. Therefore, packet matching is obviously proportional to the depth of the matching rules. Intuitively, this implies that matching can be reduced if rules that match the most significant portion of the traffic are moved to as early order as possible in the filtering policy [21]. The automation of this optimization approach is discussed in the coming sections.

3. MOTIVATION OF DYNAMIC RULE ORDERING

The behavior of Internet traffic retains several characteristics that can be exploited in the optimization of packet filters. In this section, we highlight some important Internet flow properties that we observed as a result of the traffic analysis that was performed on several Internet packet traces collected at the edge routers of DePaul University and University of Auckland networks [16]. The traces are stored as one-hour packet header logs at different days of week and times of day, where each log contains the header information for 3M to 10M packets that reflect realistic network conditions. After studying the statistics of these traffic traces, we observed the following properties pertaining to Internet flows.

Skewed flow sizes

A flow size is defined as the number of packets that constitute the flow. Referring to Figure 2(a), about 20% of the flows have 10 packets or more, and carry about 70% of the total traffic. This means that the majority of Internet traffic is constituted by a small number of heavy-weight flows. Therefore, when performing packet filtering, it is desirable to decrease the number of packet matches required for heavy-weight flows in order to reduce the overall packet matching time.

Skewed flow durations

A flow duration is defined as the time elapsed between receiving the first and last packets in the flow. Figure 2(b),

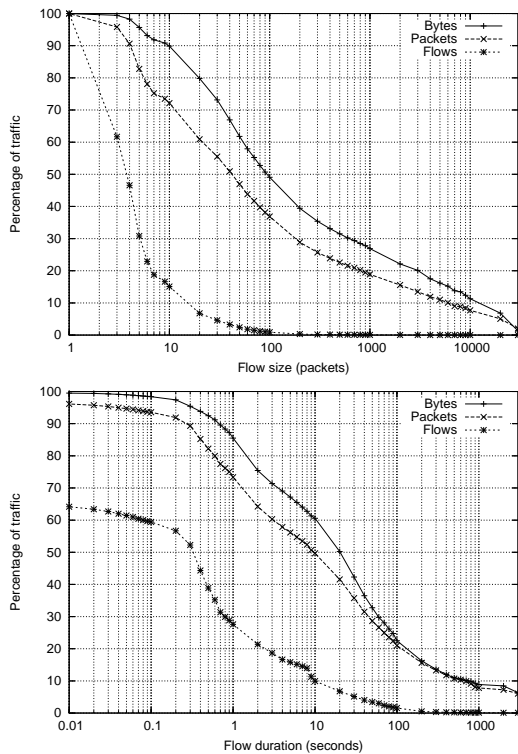


Figure 2: CCDF distribution of (a) flow size (top), and (b) flow duration (bottom) for Internet traces at the University of Auckland.

shows that about 20% of the flows last 5 seconds or more, and carry about 60% of the total traffic. This means that the majority of Internet traffic is constituted by a small number of long-lived flows. Therefore, to speed-up the packet filtering process, it is desirable to decrease the number of packet matches required for long-lived flows.

Since, in general, all packets of a flow match the same filtering rule, our observations clearly indicate that a small number of the firewall policy rules is used for matching a significant portion of the traffic over a considerable amount of time. We call this the *locality of flow matching* in firewall filtering. Previous studies [4] also support our conclusion as they show that about 90% of the packets match as little as 25% of the filtering rules in a large number of ingress routers/firewalls. This emphasizes that the idea of considering the contribution of rule matching in filtering policy optimization is useful and practical for improving the overall matching performance in firewalls.

4. OPTIMIZATION OF FIREWALL RULE ORDERING

In this section, we study the problem of optimizing the ordering of packet filtering rules to increase the performance of packet matching in firewalls. We show that the general problem is hard to solve in polynomial time, and we present a heuristic approximation algorithm that runs in polynomial time and achieves near-optimal results for the most common firewall policies.

4.1 Optimal rule ordering problem

Given a set of filtering rules with inter-rule dependencies, the optimal rule ordering problem (ORO) is to find a *legitimate* rule ordering that obtains the minimum number of packet matches. The rule ordering is legitimate only if the precedence relations between rules are preserved. Assuming that packets are sequentially matched against a policy consisting of n filtering rules with d_i as the order (depth) of rule R_i in the policy and w_i is a given weight that resembles the dominance of R_i in packet matching, we can then formally define ORO by the following the minimization problem:

$$\min \sum_{i=1}^n w_i d_i \quad (1)$$

In order to solve the ORO problem, it can be formalized as an binary integer program (BIP) as follows.

$$\min \sum_{i=1}^n \sum_{k=1}^n k w_i x_{ik} \quad (2)$$

$$\text{subject to } \forall i \sum_{k=1}^n x_{ik} = 1 \quad (3)$$

$$\forall k \sum_{i=1}^n x_{ik} = 1 \quad (4)$$

$$\sum_{k=1}^n k x_{ik} - \sum_{k=1}^n k x_{jk} < 0 \text{ if } R_i \rightarrow R_j \quad (5)$$

$$x_{ik} \in \{0, 1\}, i \in \{1, \dots, n\}, k \in \{1, \dots, n\}$$

where x_{ik} is a binary variable such that $x_{ik} = 1$ if rule R_i is positioned at location k in the policy, and $x_{ik} = 0$ otherwise. The minimization objective function (2) resembles the optimization problem described in (1) where the depth d_i of rule R_i is given by the expression $\sum_{k=1}^n k x_{ik}$ and w_i is the rule weight. The computation of rule weights is described later in Section 5.1. Constraint (3) guarantees that every rule is positioned at exactly one location, while Constraint (4) ensures that any specific location is occupied by exactly one rule. Constraint (5) preserves the ordering precedence between dependent rules by ensuring that if rule R_i has higher precedence over rule R_j (noted as $R_i \rightarrow R_j$), then the order of R_i in the policy should come before that of R_j .

The above BIP formalization allows for the use of iterative numeric techniques to solve the ORO problem [2]. A lower bound for the ORO problem can be obtained by relaxing the binary variables into linear variables and solving the BIP equations as a linear programming problem. To obtain an optimal solution, we can apply the branch-and-bound algorithm combined with gradient projection method to reduce the effect of combinatorial explosion. However, a more efficient method is needed to compute the solution since the branch-and-bound method cannot guarantee a computation time that is polynomial in the number of rules [2]. In the following section, we present our approximate heuristic solution for the ORO problem that can achieve a near-optimal solution in polynomial time.

The optimal rule ordering can be achieved when the filtering rules are disjoint by simply sorting the rules in non-increasing order based on their weights [18]. However, typ-

ical firewall policies contain dependent rules and therefore simple sorting cannot be used because ordering the rules based on their weights might conflict with rule dependencies. The ORO problem can be mapped to the job scheduling problem for a single machine with precedence constraints [6, 14]. The job scheduling problem is represented by the notation $\alpha|\beta|\gamma|\delta$, where α is the number of machines, β is the job precedence, γ is processing time restrictions, and δ is the optimization objective function. The ORO problem is similar to the $1|\beta|1|\sum_{i=1}^n w_i C_i$ scheduling problem, where w_i is the weight associated with a job and C_i is the job completion time. Since the scheduling problem was proven to be NP-Complete [13], we can prove that the ORO problem is also NP-Complete.

THEOREM 1. *The optimal firewall rule ordering (ORO) problem is NP-Complete.*

PROOF. First, we show that ORO is in NP, *i.e.*, verifiable in polynomial time. Second, we show that ORO is NP-hard by transforming the job scheduling problem $1|\beta|1|\sum_{i=1}^n w_i C_i$ to ORO in polynomial time. The transformation is performed by simply mapping jobs with precedence constraints to filtering rules with ordering precedence, and the job completion time C_i to the rule order d_i . Hence, ORO has a solution only if the job scheduling problem also has a solution. Since $1|\beta|1|\sum_{i=1}^n w_i C_i$ is NP-Complete, the ORO problem is also NP-Complete. The details of the proof is presented in [10]. \square

4.2 Heuristic ORO algorithm

Although several approximation algorithms have been proposed to solve the $1|\beta|1|\sum_{i=1}^n w_i C_i$ problem [19], the best approximation produces a solution $\left(2 - \frac{2}{n+1}\right)$ of the optimal solution, where n is the number of jobs/rules. For a reasonably large number of rules (100-1000 rules), this solution is twice as large as the optimal solution. Moreover, many of the job scheduling approximations rely on transforming the problem and solving it as a linear program, which is a significantly a complex and computationally intensive process for practical firewall deployment.

For these reasons, we developed a new heuristic approximation algorithm for the ORO problem that is more efficient and practical to implement in firewalls. The algorithm considers three common properties in real firewall filtering rules: (1) the distribution of rule weights is highly skewed such that a few number of rules match the majority of the Internet traffic, (2) the dependency depth is limited to a few number of rules, and (3) the number of dependent rules is small relative to the total number of rules in the firewall policy.

Algorithm 1 describes the details of our heuristic. The algorithm takes as an input the rule list to be optimized and an *optimization limit* that represents an upper bound on the total weight of the selected rules in the optimization process, these are called the *active rules*. It first creates a Max-Heap [11] of filtering rules based on their weights (Line 2). The Max-Heap data structure stores the item with the maximum weight on the top, such that it can be retrieved in

Algorithm 1 OptimizeActiveRules(*rule_list*, *opt_limit*)

```

1: weight  $\leftarrow$  0
2:  $H \leftarrow$  BuildMaxHeap( $H$ , rule_list)
3: while  $H$  is not empty do
4:    $R_b \leftarrow$  HeapExtractMax( $H$ )
5:   for each  $R_d \in \{\text{rules dependent on } R_b\}$  do
6:     if  $R_d \notin \text{active\_rules}$  then
7:       current  $\leftarrow$  ListTail(active_rules)
8:       while current  $\neq$  nil do
9:          $R_a \leftarrow$  ListGet(current)
10:        if Weight( $R_a$ ) < Weight( $R_d$ ) and  $R_a$  not de-
11:         pending on  $R_d$  then
12:           current  $\leftarrow$  ListPrevious(current)
13:         else
14:           break
15:         end if
16:       end while
17:       ListInsertAfter(current,  $R_d$ )
18:       weight  $\leftarrow$  weight + Weight( $R_d$ )
19:       HeapRemove( $H$ ,  $R_d$ )
20:       ListRemove(rule_list,  $R_d$ )
21:     end if
22:   end for
23:   ListInsertTail(active_rules,  $R_b$ )
24:   ListRemove(rule_list,  $R_b$ )
25:   if weight  $\geq$  opt_limit then
26:     break
27:   end if
28: end while
29: for each  $R_m \in \text{rule\_list}$  do
30:   ListInsertTail(active_rules,  $R_m$ )
31: end for
32: return active_rules

```

constant time. Rules (also called *base rules*) are sequentially picked from the heap in descending order according to their weights (Line 4). With every base rule removed from the heap, all dependent rules that must precede this rule are also removed from the heap and then inserted in the correct order in the optimized active rule list (Lines 6-17). Each dependent rule is inserted in the active rule list such that the rules in the list are in descending order according to their weights (Lines 8-12). Then, the base rule removed from the heap is appended to the optimized list (Lines 10-11). This procedure is repeated until the sum of all rule weights in the optimized list exceeds the weight optimization limit (Lines 22-24). Then, all the remaining rules in the original list are appended to the optimized active list according to their original order (Lines 26-28).

The time complexity of the algorithm is determined by two cascaded loops and the heapification operation, however, heapification is performed in $O(n \lg n)$ and can be ignored. Therefore, theoretically, the algorithm optimizes a list of n rules in $O(n^2)$ running time. However, the actual running time is only a fraction of this upper bound since every time a base rule is picked from the heap, all its dependent rules are also removed. This results in decreasing the total number of iterations in the outer while loop (Lines 3-25), and reducing the heapification time (Line 4) as well. In addition, using the optimization weight limit significantly reduces the number of iterations in the inner while loop (Lines 8-15) by

considering only the most active rules. Our study in Section 6 indicates that it is sufficient to optimize the most active 25-45% of the rules in the policy. The space complexity is obviously bounded by $O(n)$ since the algorithm keeps only two lists of filtering rules.

Notice that our algorithm uses a doubly-linked list implementation of the rule lists in order to reduce the time needed for rule insertion. Also notice that the set of rules preceding each rule in the policy can be easily accessed through a linked list of pointers created in policy pre-processing phase [10]. This pre-processing is performed only when the firewall is bootstrapped or when the filtering policy is modified.

5. IMPLEMENTATION OF DYNAMIC RULE ORDERING

In this section, we describe the implementation details of our rule order optimization technique in real firewalls. The implementation includes a method to compute filtering rule weights in order to capture the matching importance of every rule relative to others, and an adaptation mechanism that triggers the optimization algorithm only when needed based on the most recent traffic conditions.

5.1 Computation of rule weights

Each rule in the filtering policy is given a weight that reflects the dominance of this rule in matching the traffic processed by the firewall. The rule weight is calculated based on two factors: (1) *matching frequency*, which determines how frequent the rule has been triggered, and (2) *matching recency*, which determines how recent the rule has been triggered in the packet matching process.

Matching frequency

The rule frequency F_i in any time interval can be expressed as the ratio of number of packets f_i matching rule R_i to the total number of packets matched in the firewall in the same interval.

$$F_i = \frac{f_i}{\sum_{i=1}^n f_i} \quad (6)$$

Matching recency

The rule recency T_i in any time interval can be expressed as the ratio of the time t_i at which rule R_i is lastly matched to the time t_{last} of the last rule match in the firewall in the same interval.

$$T_i = \frac{t_i}{t_{last}} \quad (7)$$

Since the rule frequency and recency expressions include computationally intensive summation and floating point division operations respectively, we need to re-define these properties in a computationally simplified form in order to calculate the rule weights more efficiently. Instead of using the time as a reference, we use a packet-based virtual clock approach to measure the rule recency [28]. Simply, the number of packets received so far by the firewall is used to indicate the virtual time. When a rule matches a packet, the frequency of the rule f_i is incremented, and the current value of the global packet counter P is recorded in the rule

register p_i . For any elapsed interval of time, the frequency and recency values for rule R_i can be calculated as follows:

$$F_i = \frac{f_i}{P} \quad (8)$$

$$T_i = \frac{p_i}{P} \quad (9)$$

Rule frequency and recency are significantly important as they indicate how likely the rule will match a packet in the future as a result of the locality of matching property discussed in Section 3. Our evaluation study in Section 6 shows that the rule recency is more sensitive to long-lived bursty flows, while the rule frequency is sensitive to heavy-weight bulky flows. The weight w_i of rule R_i is a weighted average of these two factors as follows:

$$w_i = (1 - \rho)F_i + \rho T_i \quad (10)$$

The *recency ratio* ρ indicates how much of the rule weight should rely on the rule recency. Thus, the value of the recency factor is sensitive to the traffic type (bursty vs bulky). Our experiments clearly indicate that increasing the recency factor favors bursty traffic simply because bursts contain a small number of packets and last for a short period of time. The effect of the recency ratio is studied further in Section 6.

In order to avoid the division operations in calculating rule weights, we substitute with the frequency and recency values computed in Equations (8) and (9). A computationally simplified weight w'_i of rule R_i can be derived from w_i as follows:

$$\begin{aligned} w_i &= \rho \frac{p_i}{P} + (1 - \rho) \frac{f_i}{P} \\ &= \frac{1}{P} [\rho p_i + (1 - \rho) f_i] \\ w'_i &= P w_i = \rho p_i + (1 - \rho) f_i \end{aligned} \quad (11)$$

Notice that w'_i does not involve any division operations, but still measures the importance of rule R_i with respect to other rules in the policy. Therefore, w'_i can be directly used to represent rule weights in Algorithm 1 with much less processing overhead. Also notice that the weight optimization limit in the algorithm should then be multiplied by the total number of packets P .

5.2 Integration with packet matching

The optimized rule list is constructed based on the computed rule weights is used for matching upcoming packets to the firewall. The reduction in matching is maximal when the upcoming traffic distribution over filtering rules exactly matches the distribution when the list has been constructed. However, since the traffic distribution of Internet flows over filtering rules is constantly changing, the rule weights must be dynamically adjusted to reflect the most recent distribution. Therefore, we propose two types of rule list updates; performance-based triggered updates and time-based periodic updates. This achieves our goal of dynamically adjusting the rule weights to yield an ordering as close as possible to the optimal, while minimizing these updates in order to avoid excessive processing overhead.

Algorithm 2 MatchPacket(p)

```
1:  $packet\_count \leftarrow packet\_count + 1$ 
2:  $time \leftarrow \text{GetCurrentTime}()$ 
3:  $H \leftarrow \text{GetPacketHeader}(p)$ 
4:  $rule \leftarrow \text{MatchRule}(H, rule\_list)$ 
5: if  $rule \neq \text{nil}$  then
6:    $action \leftarrow rule.action$ 
7:    $rule.frequency \leftarrow rule.frequency + 1$ 
8:    $rule.recency \leftarrow packet\_count$ 
9: else
10:   $action \leftarrow \text{DEFAULT\_ACTION}$ 
11: end if
12: if  $action = \text{ALLOW}$  then
13:   $\text{ForwardPacket}(p)$ 
14: else
15:   $\text{DiscardPacket}(p)$ 
16: end if
17:  $\bar{h} \leftarrow (1 - \omega) \times \bar{h} + \omega \times rule.order$ 
18:  $\varepsilon \leftarrow K \times \bar{h} - 1$ 
19: if  $\varepsilon > \varepsilon_{thr}$ 
   or  $(time - last\_update) > \text{UPD\_PERIOD}$  then
20:   $\text{CalculateRuleWeights}(rule\_list)$ 
21:   $\text{OptimizeActiveRules}(rule\_list, \text{OPT\_THR})$ 
22:  for each  $rule \in rule\_list$  do
23:     $rule.frequency \leftarrow 0$ 
24:     $rule.recency \leftarrow 0$ 
25:  end for
26:   $packet\_count \leftarrow 0$ 
27:   $last\_update \leftarrow time$ 
28: end if
```

Performance-based triggered updates

This type of update is immediately initiated when the observed packet matching performance significantly deviates from the expected optimal matching. We use the *performance deviation factor* ε to measure the deviation of the actual average number of matches from the optimal average number of matches calculated in the last rule list update. The *update interval* is the time elapsed between two consecutive rule list update events. Thus, ε is given using the following formula:

$$\varepsilon = \frac{\sum_{i=1}^n p_i d_i}{\sum_{i=1}^n q_i d_i} - 1 \quad (12)$$

where d_i is the depth of rule R_i , p_i and q_i are the ratios of packets matching R_i in the current and preceding update intervals, respectively.

Although this formula tracks the deviation from optimal matching accurately, it is very expensive to compute at runtime for every packet received by the firewall. Therefore, we calculate the average number of packet matches so far using an exponential moving average \bar{h} as follows:

$$\bar{h}_j = (1 - \omega)\bar{h}_{j-1} + \omega h_j \quad (13)$$

where h_j is the depth of the filtering rule that matched packet j . Therefore, the performance deviation can be computed at any instance of time using \bar{h} from the above equation as follows:

$$\varepsilon = \frac{\bar{h}}{\sum_{i=1}^n q_i d_i} - 1 = K\bar{h} - 1 \quad (14)$$

where $K = 1/\sum_{i=1}^n q_i d_i$ is a constant calculated once every time the rule list is optimized. The deviation factor is computed after every packet is matched against the rule list, and if its value exceeds a certain *deviation threshold*, a new optimized rule list is constructed. The deviation threshold ε_{thr} is a user configurable parameter to specify the maximum acceptable deviation from the optimal average matching.

Time-based periodic updates

Performance-based triggered updates are important to boost the packet matching performance when it significantly drops below the deviation threshold. However, these updates are not sufficient to detect average performance deviation that is just below the threshold even if it lasts for a prolonged time interval. Thus, periodic updates are performed at fixed time intervals that are relatively long. Using the latest traffic statistics, a new active rule list is constructed using fresh rule weights in order to boost up the matching performance close to its optimum level. The update period should be determined based on the computational capacity of the filtering device.

Packet matching algorithm

Algorithm 2 describes the integration of the adaptive rule order optimization in a typical firewall packet matching module. The algorithm performs the standard packet matching procedure by matching the packet header against the rule list and performing the corresponding filtering action (Lines 2-15). As packets are received, the global packet counter is incremented and the rule frequency and recency are updated (Lines 1,6,7), and the current average number of matches and the performance deviation are computed using Equations 13 and 14 (Line 17). If the current deviation exceeds the allowable threshold or the last periodic update interval expires, the rule order optimization algorithm is invoked after calculating the new rule weights (Lines 19,21). Then, the global packet counter as well as the frequency and recency of every rule are reset to zero (Lines 22-26).

It is important to notice that the extra processing added to the packet matching algorithm imposes minor processing overhead. On one hand, the processing of every packet involves only six arithmetic operations (four assignment operations in Lines 1,2,7,8, and two simple addition and multiplication operations in Lines 17,18). On the other hand, triggered and periodic updates are performed infrequently when rule list update is required. This is shown in the results of our experiments in Section 6.

6. PERFORMANCE EVALUATION

To study the performance gain of our proposed rule order optimization technique in firewall filtering, we conducted a large set of simulation and packet trace analysis experiments. The gain in performance is measured in terms of the reduction in packet matching due to using our adaptive rule optimization mechanism as compared to the matching using the original un-optimized rule list.

6.1 Accuracy of heuristic ORO algorithm

In this section, we study the accuracy of our heuristic solution for solving the optimal rule ordering problem with different filtering policy characteristics. We conducted a num-

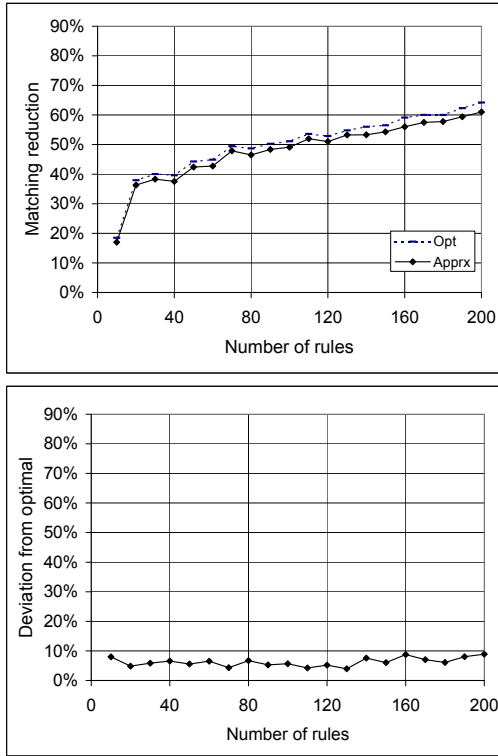


Figure 3: The effect of the number of rules on matching performance when all rules are included in the active set.

ber of simulation experiments to compare our heuristic to the optimal solution produced by solving the BIP model of the ORO problem presented in Section 4 using MatLab.

In the first experiment, we study the accuracy of our approach to optimize a varying number of filtering rules. In these scenarios, all rules are included in the active rule list (optimization limit is 100%), 20% of the rules are dependent on other rules, and each rule is related to an average of 3 preceding rules in the list, which resembles common configurations of firewall policies [23]. Rule weights are assigned based on a Zipf distribution of skewness factor 1.2 to reflect the skewed characteristics seen in the Internet traffic [25]. Figure 3 (a) shows that the reduction in matching produced by our proposed heuristic is very close to the optimal solution for rule list sizes ranging from 10 to 200 rules. Figure 3(b) shows that the deviation of our technique from the optimal solution is less than 10%.

The second set of experiments study the impact of the number of active rules on the performance of our solution. The set of active rules is chosen from a policy composed of 200 filtering rules with 20% dependency ratio and 3 average dependency depth. The experiment is repeated for different values of the skewness factor s used to calculate the rule weights based on Zipf distribution [29]. Figure 4(a) shows that for all values of s , the gain in matching increases rapidly as the number of active rules increases up to 25% of the total number of rules. After this point, the experiment shows that increasing the number of active rules im-

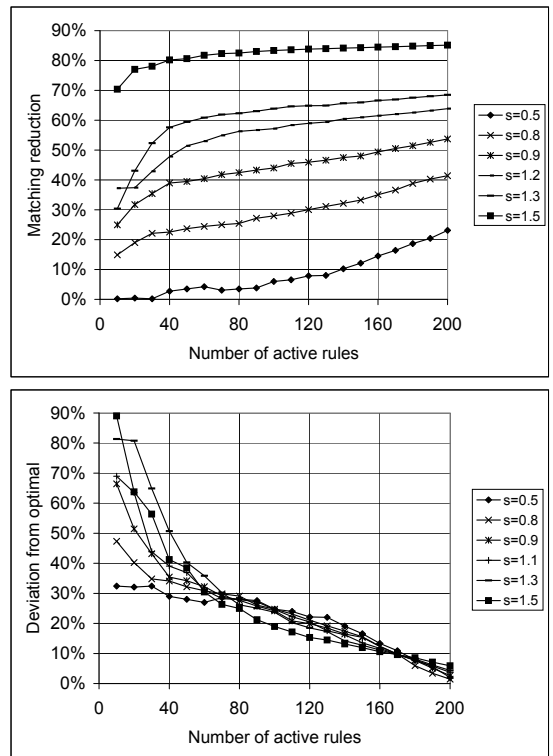


Figure 4: The effect of the active rule set size on matching performance for a firewall policy of 200 rules.

proves (though slowly) the matching gain. However, and the rate of improvement is significantly impacted by the skewness values (1.0-1.5), the matching reduction improves only by 10% when all the rules are included in the optimization. This indicates that even for a large number of firewall rules, optimizing as little as 25% of the rules can significantly improve the packet matching performance. Figure 4(b) shows that including 25% of the rules produces a solution within 30% deviation from optimal. However, more than 80% of the rules should be optimized in order to get a solution within 10% deviation.

We also evaluate how our proposed heuristic performs under various filtering policy configuration styles [27]. Figure 5 shows the performance when the average dependency depth varies from a few rules up to 50 rules for a policy composed of 200 rules with a 20% average dependency ratio. We notice that, as the rule dependency depth increases substantially, the matching gain decreases in both the optimal ordering as well as our heuristic. This is because increasing the number of rules in the dependency relation limits the possibility of finding more rule ordering solutions and thus imposes more constraints on the optimization results. This figure also shows that our heuristic gives very close results to the optimal solution when a relatively small number of dependent rules exists. For 10 or less dependent rules, our heuristics is within 5-10% deviation from the optimal solution. As the number of dependent rules increases, the deviation from optimal significantly increases. Figure 6 shows

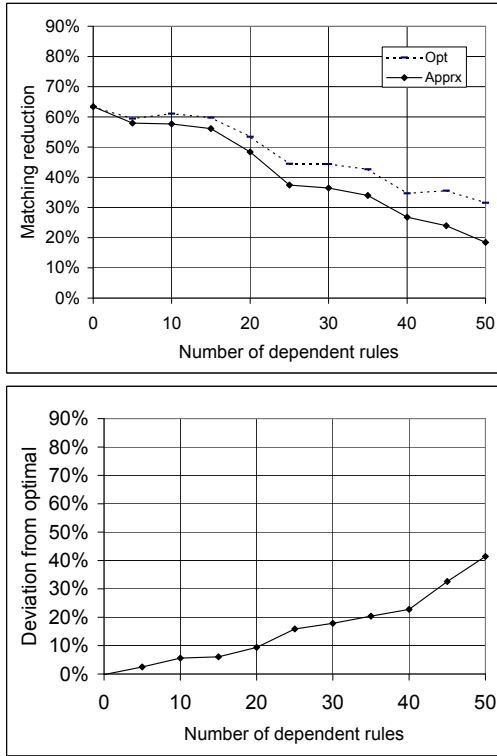


Figure 5: The effect of the average dependency depth on matching performance for a firewall policy of 200 rules.

the variation in matching performance as the average ratio of dependent rules in the same policy increases when the average dependency depth is 3 rules. Even for moderate dependency ratio values (less than 40%), our heuristic solution is as close as 5-10% from optimal. It is practically very uncommon to have firewall configurations that contain more than 40% dependent rules with more than 10 rules in the dependency relation.

6.2 Simulation and traffic trace experiment

In this section, we investigate the parameter tuning for our mechanism in order to achieve the best performance under different traffic conditions. For this purpose, we conducted many experiments using simulation as well as analyzing real Internet traffic traces. The traffic simulation experiments are conducted using the SimJava discrete event simulator [24]. Different variations of bursty and bulky traffic are generated with random inter-packet time gaps following exponential distribution [25]. The firewall implementation including rule order optimization is also simulated based on sequential rule matching. In addition, we used our firewall simulator to evaluate the optimization performance of our solution using a wide variety of Internet packet traces [16]. Based on the traffic flow information in these traces, we generated filtering rules to handle inbound traffic to the network such that the rules are created to match different field values in this traffic. The size of the generated policy (200 rules) typically resembles a reasonable rule list size in real firewall policies [27].

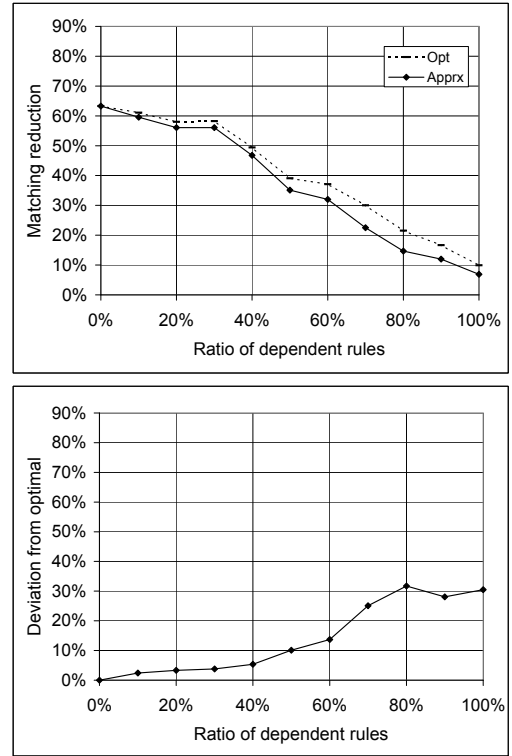


Figure 6: The effect of the average dependency ratio on matching performance for a firewall policy of 200 rules.

Effect of recency factor

In this experiment, we study the effect of the recency factor on matching performance. The results are shown in Figure 7(a). The graph shows that for bursty flows, the performance gain increases with the recency ratio (ρ), whereas for bulky flows, the gain slightly decreases. This can be intuitively explained by the fact that rule recency is more sensitive to traffic bursts that last for short time intervals. On the other hand, a high recency ratio tends to ignore bulky traffic lasting for longer time intervals. Thus, increasing the recency ratio enables the rules matching bursty traffic to get higher weight (*i.e.*, earlier order) than other rules, resulting in less matching. On the contrary, rules matching bulky traffic are pushed to a later order, increasing the number of packet matches. Figure 7(b) shows that the matching performance for the Internet traffic traces. We can observe that the matching reduction slightly decreases with increasing the recency factor similar to the simulation results for mostly bulky traffic.

The recency ratio should be chosen based on the nature of the traffic flowing in the firewall. When bursty flows are dominant, a large value of the recency factor should be used, while it should be very small for dominant bulky flows. The traffic characteristics can be easily measured using widely available traffic measuring tools/equipment like Netflow [22]. Using such a tool, the average number of bursty versus bulky flows could be determined, and the corresponding recency ratio values can be retrieved from lookup tables constructed using simulated traffic. From our experiment, a 0.2% re-

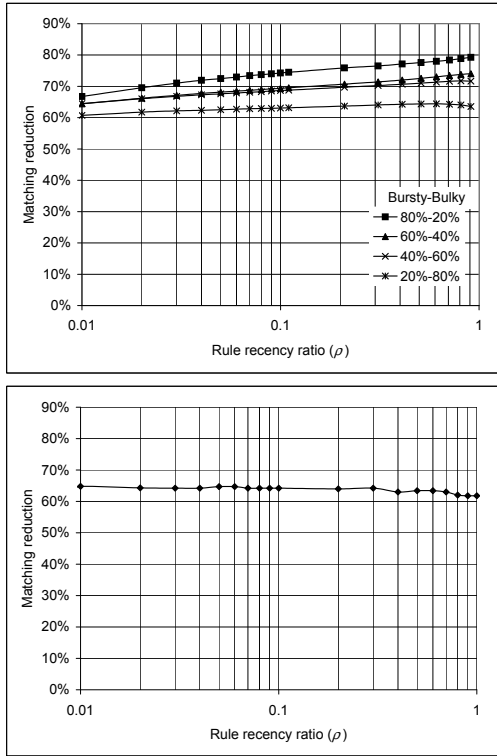


Figure 7: The effect of the recency ratio (ρ) on matching performance for (a) simulated traffic (top) and (b) Internet traffic traces (bottom).

recency ratio achieves matching gain close to the best possible value for both bursty and bulky flows.

Effect of optimization weight limit

In this experiment, we study the effect of the optimization weight limit on matching performance. The results are shown in Figure 8(a). The graph shows that, for bursty flows, the matching reduction increases rapidly with increasing the optimization weight limit up to a plateau value where the matching reduction becomes almost constant. The matching reduction reaches its highest value when the active rule list includes about 90% of the total rule weights. For a typical skewed rule weight distribution, a weight limit of 90% includes less than 45% of the rules in the policy. The results for the same experiment on Internet traffic traces are shown in Figure 8(b). The graph reflects similar results, with a matching reduction plateau at a weight limit of 80%, corresponding to about 25% of the rules. These results clearly show that using an optimization weight limit of about 80-90% significantly reduces the number of rules added to the active list, which consequently reduces the corresponding processing overhead needed to optimize these rules.

Tuning of triggered updates

In this experiment, we closely examine the dynamics of the adaptive rule list update mechanism during a one hour interval on a weekday from 12:00pm to 12:59pm. The performance deviation is calculated periodically every 20 seconds based on the formulas in Section 5.2. The active rule list

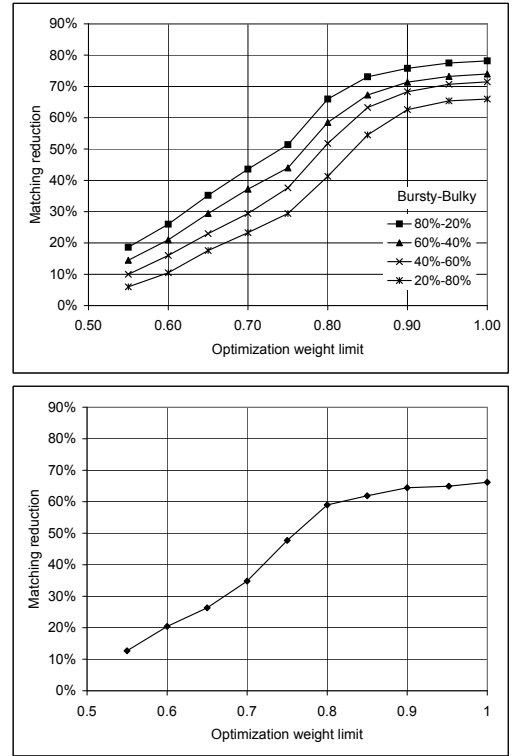


Figure 8: The effect of the optimization weight limit on matching performance for (a) simulated traffic (top) and (b) Internet traffic traces (bottom).

is dynamically reconstructed whenever the average number of matches experience more than 10% deviation from the optimal value. Figure 9 shows the results of this experiment with list updates indicated by the solid triangular marks.

The graph shows that the performance deviation changes smoothly with every packet received, thus ignoring sudden short-term decrease in the instantaneous matching performance. When the matching performance trend sustains a continuous decline, the deviation factor exceeds the designated threshold and the optimized list is reconstructed based on the most recent rule weights. Optimizing the rule list is computationally intensive and should be performed only when crucially needed. The frequency of rule list updates is tightly coupled with the deviation threshold ε_{thr} and the averaging smoothing factor ω . Our study of various settings of these parameters shows that, during network rush hours, our adaptive technique reconstructs the list only 2-5 times in an hour with setting $\omega = 0.4$ and $\varepsilon_{thr} = 0.1$. This incurs minor amortized overhead throughout the full interval in which the optimization algorithm is utilized.

Long-term optimization performance

Figure 10 shows the performance of our technique 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 constant level for a long period of time. The maximum gain (around 60%) is achieved during day hours where fast filtering is highly needed. This is attributed to the existence of a large traffic volume that

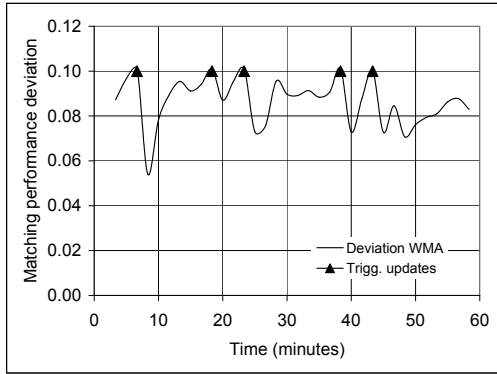


Figure 9: WMA of performance deviation between 12:00pm and 12:59pm.

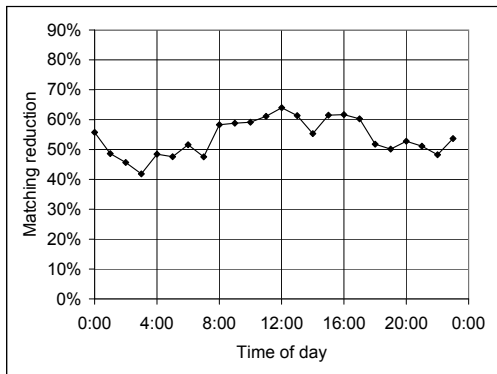


Figure 10: Hourly-average optimization performance throughout a weekday.

consequently creates significant skewness in the rule matching statistics and consequently rule weights. During evening and night hours, the traffic volume is much less, hence creating a reduced degree of skewness decreasing the gain in the relative matching reduction.

7. RELATED WORK

The optimization of packet filters have been extensively studied in recent research. The main solutions to improve the packet matching time use various combinations of one or more of the following: hardware-based solutions, specialized data structures, geometric algorithms, heuristics and statistical techniques.

Hardware-based solutions using Content Addressable Memories (CAM) [15] 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. A wide variety of specialized data structures have been used for fast packet filtering. Tuple space search [20] builds a table of all possible field value combinations and pre-compute the earliest rule matching each combination. Geometric algorithms map the values of filtering fields to ranges in geometric dimensions. Fat Inverted Segment (FIS) trees [5] are used to partition these dimensions into a number of regions.

Decision-tree based classification algorithms [8, 26] build a decision tree using local optimization decisions at each node to choose the next bit or field to test. Heuristic approaches like Recursive Flow Classification (RFC) [7] pipeline various packet matching stages to achieve high throughput in hardware implementations. Although all these works contribute significantly to the advancement of packet filtering 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, which limits their practical deployment in filtering devices.

The idea of re-ordering filtering rules based on traffic behavior in order to reduce packet matching has been addressed by network device vendors [21]. However, the available documentation only provides general guidelines for administrators to manually position the rules that match large traffic portions ahead of other rules in the policy. The work in [17] presents algorithms to optimize filtering policies by aggregating adjacent rules and eliminating redundant ones to reduce the size of the rule list. However, the work did not use any traffic information in the technique. Statistical data structures are also used in optimizing packet filtering [9]. 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. The work in [4] investigates the problem of rule order optimization for packet filters, and presents the outline of a greedy approximation algorithm. However, the algorithm produces close to optimal solution only in the case when the majority of the rules are disjoint. In addition, the paper provides no further details on how rule weights are computed based on collected traffic statistics, how these weights are dynamically updated with the current traffic conditions, and how the optimization algorithm can be seamlessly integrated in the packet matching module of filtering devices.

8. CONCLUSION

The packet filtering 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 presents a new approach for optimizing packet filtering in network security policies based on online traffic statistics. The paper presents the techniques, algorithms and evaluation study to tackle the problem effectively.

We first show that the matching skewness of firewall rules is a profound property to utilize in statistical packet matching. We use this property to optimize the ordering of filtering rules such that it adapts to changes in the traffic characteristics with minimal processing overhead. We consider two factors to determine the importance of a rule (rule weight) in the rule ordering algorithm: rule frequency and recency, which reflect the number and time of rule matching, respectively. We also show how these statistics are calcu-

lated from the network traffic with minimal computational and space overhead. The space complexity of our algorithm is bounded by $O(n)$, and the computational complexity is shown to be $O(n^2)$. We can argue that using statistical optimization of firewall rule lists guarantees obtaining minimal average matching time when the traffic statistics get stable over time. Our evaluation of the proposed approach using Internet traffic traces shows that our technique achieves 60% average matching reduction during day hours when high performance is crucially needed. We also developed an adaptive mechanism to update the rule list dynamically and keep the matching performance close to optimal with minimum packet matching interruption (about 2-5 times/hour). The implementation of our mechanism is simple, lightweight and introduces minimal processing overhead on the standard packet filter processing.

9. REFERENCES

- [1] E. Al-Shaer and H. Hamed. Modeling and management of firewall policies. *IEEE Transactions on Network and Service Management*, 1(1), April 2004.
- [2] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.
- [3] D. Chapman and E. Zwicky. *Building Internet Firewalls*. Orielly & Associates Inc., second edition edition, 2000.
- [4] E. Cohen and C. Lund. Packet classification in large ISPs: Design and evaluation of decision tree classifiers. *ACM SIGMETRICS Performance Evaluation Review*, 33(1):73–84, 2005.
- [5] A. Feldmann and S. Muthukrishnan. Tradeoffs for packet classification. In *IEEE INFOCOM'00*, March 2000.
- [6] R. Graham, E. Lawler, J. Lenstra, and A. Kan. Optimizing and application in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5, 1979.
- [7] P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, 15(2):24–32, 2001.
- [8] P. Gupta and N. McKeown. Packet classification using hierarchical intelligent cuttings. In *Interconnects VII*, August 1999.
- [9] P. Gupta, B. Prabhakar, and S. Boyd. Near optimal routing lookups with bounded worst case performance. In *IEEE INFOCOM'00*, 2000.
- [10] H. Hamed and E. Al-Shaer. Adaptive statistical optimization techniques for firewall packet filtering. Technical Report TR-05-012, DePaul University, 2005.
- [11] D. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, Reading, Massachusetts, third edition.
- [12] K. Lan and J. Heidemann. On the correlation of internet flow characteristics. Technical Report ISI-TR-574, USC/ISI, 2003.
- [13] E. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics*, 2, 1978.
- [14] J. Lenstra and A. Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1), 1978.
- [15] A. J. McAulay and P. Francis. Fast routing table lookup using CAMs. In *IEEE INFOCOM'93*, March 1993.
- [16] Passive Measurement and Analysis Project, National Laboratory for Applied Network Research. Auckland-VIII Traces. <http://pma.nlanr.net/Special/auck8.html>, December 2003.
- [17] J. Qian, S. Hinrichs, and K. Nahrstedt. ACLA: A framework for access control list (ACL) analysis and optimization. In *IFIP Communications and Multimedia Security*, 2001.
- [18] R. Rivest. On self-organizing sequential search heuristics. *Communications of the ACM*, 19(2), 1976.
- [19] A. Schulz. Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds. In *The 5th International IPCO Conference*, 1996.
- [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] Cisco Systems. Optimizing ACLs. User Guide for ACL Manager 1.4, CiscoWorks2000, 2002.
- [22] Cisco Systems. Netflow services solutions guide, October 2004.
- [23] D. Taylor and J. Turner. Scalable packet classification using distributed crossproducting of field labels. In *IEEE INFOCOM*, 2005.
- [24] SimJava v2.0. Process based discrete event simulation package for java. <http://www.dcs.ed.ac.uk/home/hase/simjava/>, 2002.
- [25] J. Wallerich, H. Dreger, A. Feldmann, B. Krishnamurthy, and W. Willinger. A methodology for studying persistency aspects of internet flows. *SIGCOMM Computer Communication Review*, 35(2), 2005.
- [26] Thomas Y. C. Woo. A modular approach to packet classification: Algorithms and results. In *IEEE INFOCOM'00*, pages 1213–1222, March 2000.
- [27] A. Wool. A quantitative study of firewall configuration errors. *IEEE Computer*, 37(6):62–67, 2004.
- [28] L. Zhang. Virtual clock: a new traffic control algorithm for packet switching networks. In *The ACM symposium on Communications Architectures and Protocols*, 1990.
- [29] G. Zipf. *Human Behaviour and the Principle of Least-Effort*. Addison-Wesley, 1949.