Dynamic Ordering of Firewall Rules Using a Novel Swapping Window-based Paradigm

Ratish Mohan*
Computer Science department
University College of Oslo and Akershus
Oslo, Norway

Anis Yazidi†
Computer Science department
University College of Oslo and Akershus
Oslo, Norway
anis.yazidi@hioa.no

Boning Feng‡
Computer Science department
University College of Oslo and Akershus
Oslo, Norway
boning.feng@hioa.no

B. John Oommen§
School of Computer Science
Carleton University, Ottawa, Canada
oommen@scs.carleton.ca

ABSTRACT

Designing and implementing efficient firewall strategies in the age of the Internet of Things (IoT) is far from trivial. This is because, as time proceeds, an increasing number of devices will be connected, accessed and controlled on the Internet. Additionally, an ever-increasingly amount of sensitive information will be stored on various networks. A good and efficient firewall strategy will attempt to secure this information, and to also manage the large amount of inevitable network traffic that these devices create. The goal of this paper is to propose a framework for designing optimized firewalls for the IoT.

This paper deals with two fundamental challenges/problems encountered in such firewalls. The first problem is associated with the so-called "Rule Matching" (RM) time problem. In this regard, we propose a simple condition for performing the swapping of the firewall's rules, and by satisfying this condition, we can guarantee that apart from preserving the firewall's consistency *and* integrity, we can also ensure a greedy reduction in the matching time. It turns out that though our proposed novel solution is relatively simple, it can be perceived to be a generalization of the algorithm proposed by Fulp [1]. However, as opposed to Fulp's solution, our swapping condition considers rules that are not necessarily consecutive. It rather invokes a novel concept that we refer to as the "swapping window".

The second contribution of our paper is a novel "batch"-based traffic estimator that provides network statistics to the firewall placement optimizer. The traffic estimator is a subtle but modified batch-based embodiment of the Stochastic Learning Weak Estimator (SLWE) proposed by Oommen and Rueda [2].

The paper contains the formal properties of this estimator. Further, by performing a rigorous suite of experiments, we demonstrate that both algorithms are capable of optimizing the constraints imposed for obtaining an efficient firewall.

CCS Concepts

•**Security and privacy** → **Firewalls;** •**Theory of computation** → **Online learning theory;** •**Networks** → *Network experimentation;*

*Former Graduate Student
†Associate Professor
‡Associate Professor
§Fourth author status: *Chancellor's Professor*, *Fellow: IEEE* and *Fellow: IAPR*. The author is also an Adjunct Professor with University of Agder, Grimstad, Norway.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Keywords

Firewall Optimization, Matching time, Weak Estimators, Learning Automata, Non-Stationary Environments, Batch Update

1. INTRODUCTION

The inter-connectivity, convenience and the all-prevalent digital services offered by the Internet, come with a steep price. As our society becomes more dependent on the Internet, the requirement to secure the information stored on these devices and services is more stringent and demanding. To secure the information, the users and systems' administrators have to be even more security-conscious.

The filed of computer security is extensive. It encompasses the security of the physical machines as well as the information stored on them. However, in the context of the Internet, one has to be additionally concerned with *network*-related aspects of security. Such a "specialized" form of security is mandatory especially because an increasing number of devices are connected to various networks, and primarily to the Internet [3]. Network security deals with the security aspects of data and communication within a/multiple network(s), and it spans many different concepts such as authentication, access policies, intrusion detection, intrusion prevention and honeypots/honeynets.

A first line of defence in network security is to use a firewall in order to enforce access policies. A firewall, in essence, is a system architecture program whose objective is to filter the incoming and outgoing packet traffic on a host or in a network. The task of accepting or denying access to the network is enforced by matching the header information of each data packet against a predefined set of rules, referred to as the "firewall policy". Each rule has an action associated with it, for example, to either deny or accept access, and this action is what decides whether a packet is dropped or not.

A study of many Internet and private traces shows that the major portion of any network's traffic matches only a small subset of firewall rules. This, in turn, implies that the frequency distribution for some of the traffic properties appears to be highly skewed [4]. Furthermore, when performing packet filtering, each rule in a firewall policy will usually be checked in a sequential order. Consequently, as the firewall policy increases in size, as any rule is often combined with a matching rule of a higher order, the overhead associated with the task of filtering the firewall, will become increasingly costly.

The reader will easily see that this rule matching phase can easily become a bottleneck in a high speed network when it is under attack or when it encounters a heavy network load [1, 5]. Furthermore, it is well known that the computing power of hosts, the transmission speeds of packets and the complexity of networks, continue to increase. To keep abreast with these increasingly-demanding environments, firewalls must be able to "proportionately" adapt to changes by processing packets at increasingly higher speeds [6, 7]. Thus, it is desirable that a firewall monitoring system processes a lesser number of packet matches in order to reduce the potentially exorbitant filtering overhead as well as the overall packet matching time [4].

A natural inference of the above assertions is the following: In order reduce the number of packet matches that have to be processed, and to ensure that a firewall is able to process packets at an adequate speed, it is crucial for the firewall's architecture to have an optimized *ordering* for the appropriate rules. This can be achieved by ensuring that the rule ordering is such that the rules that are matched most often, appear at the top (front) of the list of rules. This will reduce the amount of time used to process a packet by reducing the number of required packet matches, and consequently reducing the packet filtering overhead. Additionally, it will also have the effect of improving the network's throughput because a packet will spend less time being processed.

Although the problem is easily stated, the task of finding the optimal rule order is NP-hard because of inter-rule dependencies. Our goal is thus to find a heuristic algorithm to find a near-optimal rule order.

The complexity of the problem is accentuated by the fact that traffic patterns in networks are not static. This implies that since the patterns are dynamic and possibly time varying, one cannot learn the statistics of the traffic patterns using traditional estimation methods. Rather, one has to devise estimation (or learning) strategies that are rather effective for non-stationary environments. This is the task we undertake!

1.1 Problem Statement and Contributions

Put in a nutshell, this paper deals with *dynamic networks*, i.e., those that are characterized by being "under constant change and activity". Essentially, a dynamic network is one in which the state of packet traffic is time-varying and non-stationary. This implies that the packet traffic fluctuates in such a way that no single type of traffic is dominant for an extended period of time. Our goal is to find a solution to the problem of optimizing the *performance* of a firewall in such dynamic networks. In order to achieve this, we attempt to answer the following questions:

- *How can we optimize the order of the firewall rules in order to minimize the Rule Matching (RM) operations invoked?*

- *How can we learn and use the dynamic network traffic statistics to further optimize the firewall?*

Of course, to achieve the above, we shall examine the traffic patterns statistically, combine the inferences with the workings of a RM algorithm. Thus, the major contributions of this paper are:

- We present an efficient and yet simple mechanism for optimizing the order of the rules in the firewall by using a novel concept that is referred to as the **Swapping Window**. The **Swapping Window** is a straightforward strategy by which one can infer whether it is beneficial to swap the *order* of two rules in a RM algorithm by considering their matching probabilities, and simultaneously guaranteeing that no inconsistencies are introduced in the firewall. We submit that, without loss of generality, our solution is a mapped efficient solution to the ***Single Machine Job Scheduling*** (SMJS) Problem [8] – since our problem can be shown to be a specific instantiation of the latter.
- We present a novel adaptive algorithm for estimating the statistics of multinomial observations appearing in a batch mode[1]. The algorithm is able to deal with non-stationary environments and is an extension of the Stochastic Learning Weak Estimation (SLWE) work by Oommen and Rueda [2], which is, in and of itself, suitably adapted for high speed networks. The observations that the estimation scheme receives are, in our case, the different matched rules within a time interval when they are examined as a "batched" data stream and not as sequential entities.
- We combine both the above-mentioned contributions (the rule ordering algorithm augmented with the estimation scheme) into a single algorithm so as to achieve a holistic approach for optimizing the firewall's performance.

1.2 Organization of the Paper

After having introduced and motivated the problem in Section 1, we proceed to review the related state-of-the-art in Section 2. In Section 3, we present our solution composed of two main components: Rule Re-ordering (RR) and traffic estimation.

In Section 4, we present some theoretical results that demonstrate the validity of the algorithms proposed in Section 3 for both rule ordering and estimation. Section 5 contains simulation results demonstrating the power of the scheme in dynamic environments. The experiments done for dynamic environments were based on a realistic test-bed.

Section 6 concludes the paper.

[1]The batched-mode version of Oommen-Rueda's SLWE is a contribution in its own right to the field of estimation in non-stationary environments.

2. STATE-OF-THE-ART

This section outlines the current state-of-the-art when it concerns firewall optimization.

2.1 Firewall Matching Optimization: Legacy Approaches

In this section, we describe the relevant rule order optimization algorithms that are based on matching optimization, and outline the general problem of firewall Rule Reordering (RR).

The task of optimizing a firewall is comparable to that of solving the ***Traveling Salesman Problem*** (TSP) [9] with precedence constraints [8]. The standard TSP is defined as the task of finding the shortest route while traversing each city exactly once, given N cities and their intermediate distances. However, as observed by the author of [10], when constraints are included in the problem, it becomes more complex. The authors of [8] examined a variant of the TSP with precedence constraints. This variant, known as the Time-Dependent Traveling Salesman Problem (TDTSP), considers the case when transition costs between two cities depends on the time of the visit[2]. This implies that certain cities can only be visited at a given time, and thus, trying to find an optimal path with such a constraint means that some cities must be visited before others due to the dependency relationships between the cites. This is precisely, a mapping of the problem of finding the optimal rule ordering in a firewall policy with dependency relationships, because finding the optimal rule ordering in a firewall entails creating a rule ordering such that some rules must be "visited" or compared against before other rules, until a match is found.

2.1.1 A Bubble Sort-like Algorithm

Since the problem is NP-hard, the author of [1], designed a simple heuristic algorithm, given in Algorithm 1, for optimizing firewalls rule ordering.

[2]The authors of [8] confirm that the TDTSP can be mapped onto ***single machine job scheduling problem*** [8] which is known to be NP-hard [11]. Thus the optimization problem for firewall rules is also NP-hard. The only option to find the optimal solution requires an exhaustive search of the solution space — which is not a scalable solution. Rather, one must be content to find a sub-optimal solution using a heuristic algorithm.

Algorithm 1: A simple Bubble Sort-like rule ordering algorithm.

```
Data: A list of firewall rules
Result: A new and improved ordering of firewall rules
1 done = False
2 while !done do
3     done = True
4     for (i = 1; i < n; i++) do
5         if (p_i < p_{i+1} AND r_i ∩ r_{i+1} = ∅) then
6             interchange rules and probabilities
7             done = False
```

By studying the algorithm, one observes that it is similar to the Bubble Sort algorithm. It compares neighbours and, if possible, swaps them. Further [1], in order to preserve the rule precedence relationships, the algorithm uses rule probabilities and rule intersection as the swapping criteria. For example, consider the scenario when there are two rules, i.e., Rule1 and Rule2 where Rule1 has a lower probability than Rule 2, and where the rules don't intersect. This means that the rules are not dependant on each other and are thus "swappable". The algorithm will process the rules, in a pair-wise manner, until there are no more "swappable" rules.

The problem with this algorithm, however, is that one rule can prevent another from being re-ordered [1], rendering the algorithm to be unable to re-order groups of rules. The following is an example of this problem; suppose there are three rules, namely Rule1, Rule2, and Rule3. Rule 1 and Rule 3 have a dependency relationship, and the rules have the associated probabilities given in Table 1:

Rule	Prob.
Rule1	0.1
Rule2	0.5
Rule3	0.4

Table 1: An example with a small number of rules with their probabilities.

Ideally, the rule with the highest matching probability would appear at the beginning of the list of rules in order to reduce the number of packet matches. Thus, in order to preserve the dependency relationships, the optimal rule order is: Rule1, Rule3, Rule2. However, the algorithm by [1] is not able to achieve this rule ordering, as explained below.

The algorithm will first swap Rule1 with Rule2. It will then check if Rule1 can be swapped with Rule3, but because they intersect, they will not be swaped. In the second iteration of the While loop the problem encountered becomes evident. Indeed, because Rule2 is better than Rule1 they will not be swapped, and further, because Rule1 and Rule3 intersect they will not be swapped either. Thus, when the algorithm terminates, the final order will be, clearly, suboptimal[3].

However, despite its problems, this algorithm will still create a rule ordering that is better than the original, if possible.

2.1.2 A DAG-based Algorithm

The authors of [12] presented a heuristic algorithm for optimized policy RR that is able to re-order a policy containing precedence relationships (or a sub-graph in the **DAG**) in such a way that the policy integrity is maintained. A short synopsis of the most important aspects of this algorithm is given below.

The algorithm functions by operating on certain data structures. It needs a set, $G(r_i)$, of rules containing the sub-graph rules of r_i, i.e. the dependency relationships for r_i. It also uses a FIFO Queue, S, to represent the optimal policy rule sorting, where S is initially empty. Additionally, it requires a list, Q, containing the rules to be sorted, and this list is initially equal to the original firewall policy, R.

For each pass, the algorithm selects the rule with the highest average subgraph probability from the graph of rules available during *that particular* pass. The selected rule is then inserted into the list of sorted rules, S, if it has no rules dependent on it. Otherwise, the algorithm iteratively sorts the subgraph of its dependents until it finds a rule that has no dependent rules and inserts that rule into the list of sorted rules. The algorithm then updates the respective data structures and repeats the process until all the rules have been placed in S.

3. PROPOSED SOLUTION

3.1 Overview of the Solution

The goal of this paper, as expressed in the problem statement, is not merely to create a rule ordering algorithm. Rather, our aim is to explore the problem of optimizing a firewall's **performance** in a **dynamic network**. This means that for the firewall to have an optimised performance at all times, there needs to be an explicit understanding of when the rules have to be re-ordered as the network traffic dynamically begins to favour other rules in the policy.

This implies the need for two algorithms: The first algorithm must be useful to achieve the necessary RR, and the second one must be capable of updating the rule probabilities as the network traffic fluctuates. From an overall perspective, we also need a single scheme that connects both

[3]This is a very simplistic example. Indeed, the possibility of terminating on suboptimal solutions is accentuated when the number of rules is larger.

these algorithms into a single, optimized adaptive firewall. We first consider the requirements for both these algorithms.

- **The RR algorithm** should to be able to sort a firewall's rule order based on each rule's matching probability, dependency relationships, and firewall position. This will ensure that the average packet matching time is reduced. In order to satisfy these criteria, the algorithm will need to have access to the current firewall security policy, a knowledge of the dependency relationships, and the matching probabilities of every rule. The details of this algorithm are presented in Section 3.2.
- **The traffic aware algorithm** should be able to update a rule's matching probability dynamically as the network's traffic state changes. This means that this algorithm will need to have access to the currently-applied firewall security policy and the current number of packet matches for each rule. In order to enable dynamic estimation of the rule matching probabilities, we present a novel weak estimator, which is a central component of our approach, in Section 4.1.
- Finally, the above two algorithms must be combined in such a way that they can communicate with each other. The traffic aware algorithm needs to be able to update the probability associated with a rule, and this update must be reflected in the rules used by the RR algorithm. If this is not achieved, the RR will never be able to find the optimal rule ordering of the firewall when the traffic state of the network changes. Thus, we will, briefly, describe two mechanisms for triggering the RR, namely, periodically and "performance triggered". These are described briefly in Section 4.2, and in more detail in the section that reports the experimental results that we have obtained, Section 5.

The primary reason why the problem is complex is because the RR and traffic-aware criteria themselves may be conflicting. Indeed, rule r_i may have to precede r_j with regard to the network's security policy requirements, and yet the probability of r_j being applied may be greater than that of r_i being applied. However, we will consider the RR issue first.

3.2 The Rule Re-Ordering Algorithm

The algorithm that we propose for RR, uses as its foundation the simple RR algorithm described earlier and presented in [1], namely Algorithm 1. Our new strategy is shown in Algorithm 2. However, before we formally present the algorithm, we shall explain its rationale.

3.2.1 Rationale for the Algorithm

Quite naturally, the algorithm itself takes as its input, a list of rules. It also has a list of rules that each given rule must *precede*, and which each rule must *succeed*. If these lists collectively formed a DAG that represented a *single string* of connected edges with a single source and a single sink, the problem of re-ordering the rules would have been trivial. The problem is necessarily complex because the set of lists of *preceding* and *succeeding* nodes could be potentially conflicting. Our solution represents a heuristic scheme by which these conflicting requirements are resolved in the best possible manner.

To be more specific, the algorithm itself takes as its input, a list of rules. It also maintains two data structures.

- **The preceding list** of a rule, r_i, contains all the rules that are dependent on r_i. Essentially, this means that r_i must appear **before** the rules in the preceding list in order to maintain the integrity of the policy.
- **The succeeding list** of a rule, r_i, contains all the rules that r_i is dependent on. Analogous to the above, this means that r_i must appear **after** the rules in the succeeding list in order to maintain the policy's integrity.

Our algorithm contains two main loops that it iterates through. For every iteration of the outer loop, the inner loop will traverse the whole list. The reason for this is that the algorithm will compare the current element in the outer loop, r_x, with the current element in the inner loop, r_y.

The algorithm will then try to find a **swapping window** between r_x and r_y. A swapping window is defined as an interval of positions in a firewall in which the two comparing rules can be swapped, without breaking the integrity of the firewall policy. The window is found by analyzing the two comparing rule's succeeding and preceding lists.

By finding the rule with the *highest* position in the firewall in the preceding list for r_x and the rule with the *lowest* position in the firewall in the succeeding list of r_y, an interval of positions can be found. Once such an interval has been determined, the algorithm will check if the window is a valid swapping window for the current rules being compared.

In order to check the validity of the swapping window, the algorithm will check if the current position of r_x is less than the lowest position in the succeeding list of r_y and if the position of r_y is grater than the highest position in the preceding list of r_x. If the latter expression is evaluated to be *True*, the swapping window is deemed to be valid.

However, the above is only valid if r_x has a higher position in the firewall than r_y. In the case where r_y has a higher

position in the firewall than r_x (as seen in lines 12 and 13 in Algorithm 2) there is a slight difference in the swapping criteria. In this case, the r_x and r_y values in the "*If expression*" switch places. The swapping mechanism is illustrated in Figure 1.

Once the algorithm has found a valid swapping window and thus knows that r_x and r_y can be swapped without violating the integrity of the policy, it will do a simple comparison of the rules' matching probabilities in order to decide whether they should be swapped or not. Even if the algorithm determines that they should be swapped, the algorithm will not properly swap them yet. Rather, the algorithm will do this based on a criterion value, Δ_{new}, explained below.

The value Δ_{new} is created using the matching probability and position number of the rules being compared against and simply yields the estimated average matching time before and after swapping r_x and r_y. This can be said to represent the swapping rank of r_y. The higher the swapping rank, the more optimal the swap is considered to be. Consequently, the algorithm will then perform a test to check whether this Δ_{new} value is greater than the current maximum value of Δ, i.e., Δ_{max}. If it is greater, then Δ_{max} is re-set to assume this rule's Δ_{new} value, and this rule is now the optimal swapping option.

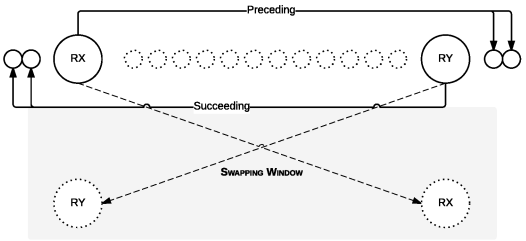

Figure 1: How Algorithm 2 re-orders rules.

When the inner loop has finished its traversal, a check is performed in order to find if r_x should be swapped with a rule or not. If it should be swapped, the rule with highest delta value, Δ_{max}, is chosen to be the optimal rule for it to be swapped with. Finally, the outer loop will complete the iteration and move on to the next rule at which point the process above is repeated for that rule.

In essence, what this heuristic algorithm tries to achieve, is to get as many rules as possible, with a high matching probability, as close to the top of the firewall as possible.

The formal algorithm follows.

Algorithm 2: Our newly-proposed Rule Re-ordering algorithm.

Data: A list of firewall rules
Result: A new and improved ordering of firewall rules

```
1  for rx in rules do
2      Δ_max = 0
3      for ry in rules do
4          Δ_new = 0
5          if r_x ≠ ry then
6              if r_x.pos < r_y.pos then
7                  if r_x.pos < succeeding_max(r_y) AND
                   r_y.pos > preceding_min(rx) then
8                      if r_x.prob < r_y.prob then
9                          Δ_new = (r_y.prob − r_x.prob) ∗
                           (r_y.pos − r_x.pos)
10                         if Δ_max < Δ_new then
11                             Δ_max = Δ_new
12             else
13                 if r_y.pos < succeeding_max(rx) AND
                   r_x.pos > preceding_min(r_y) then
14                     if r_y.prob < r_x.prob then
15                         Δ_new = (r_x.prob − r_y.prob) ∗
                           (r_x.pos − r_y.pos)
16                         if Δ_max < Δ_new then
17                             Δ_max = Δ_new
18     if Δ_max > 0 then
19         swap(r_x, r_y)
```

4. THEORETICAL RESULTS: ESTIMATION AND RULE ORDERING

4.1 Designing a Weak Estimator for Batch Updates

Having described our RR algorithm, we now proceed to the issue of traffic estimation, and design a Weak Estimator scheme that is relevant for batch updates. The algorithm that we propose is a modified version of the *weak estimator* algorithm initially proposed by Oommen *et al* [13]. It is modified in such a way that it is able to use a batch of packet matches (as opposed to a single packet match as the SLWE scheme from [13] would do) in order to calculate the packet matching probabilities for a given rule. This ensures that the algorithm does not have to constantly perform estimate updates for each incoming packet.

The algorithm takes as its input a list of rules and a value for its parameter, λ. It then iterates through the list of rules and updates the probability associated with each rule by using the modified weak estimator algorithm given below. Quite simply put, in order to update the probability associated with each rule, the algorithm calculates it using the previous prob-

ability of the given rule, $\hat{p_i}$, the total number of packet matches, M, and the number of packet matches for any single rule, m_i. The pseudocode is given in Algorithm 3.

Algorithm 3: The Weak Estimator algorithm.

Data: A list of firewall rules, and a lambda value
Result: Updated probabilities for each rule in the list of rules

1 **for** *rule i in rules* **do**
2 $\hat{p_i} = \frac{m_i}{M}\hat{p_i} + \lambda(\hat{p_i} - \frac{m_i}{M})$

4.1.1 Theoretical Results: The Batch-oriented Weak Estimator

In this section, we present some theoretical results related to our algorithms. The first result concerned the optimality of the devised Batch-oriented Weak Estimator (Algorithm 3) described above. The algorithm is a generalisation of the Stochastic Learning Weak Estimator (SLWE) proposed by Oommen and Rueda [13]. The main difference is that the Stochastic Learning Weak Estimator operates in an incremental manner, i.e., updates the estimates of the probabilities upon receiving every single observation. As opposed to this, the Batch-oriented Weak Estimator proposed here is able to handle a batch of M observations.

Specifically, let X be a multinomially distributed random variable, which takes on the values from the set $\{'1', \ldots, 'r'\}$. We assume that X is governed by the distribution $S = [s_1, \ldots, s_r]^T$ as follows:

$X =$ 'i' with the probability s_i, where $\sum_{i=1}^{r} s_i = 1$.

We assume that between two discrete time instants n and $n+1$, we obtain a batch of M concrete realisations of X. Let $\{x(n,1), x(n,2), x(n,3), ..., x(n,M)\}$ denote the batch of M observations obtained between the time instants n and $n+1$. The intention of the exercise is to estimate S, i.e., s_i for $i = 1, \ldots, r$ based on the batch of observations. We achieve this by maintaining a running estimate $P(n) = [p_1(n), \ldots, p_r(n)]^T$ of S, where $p_i(n)$ is the estimate of s_i at time 'n', for $i = 1, \ldots, r$. We omit the reference to time 'n' in $P(n)$ whenever there is no confusion.

Let $m_i(n)$ be the number of elements in the batch: $\{x(n,1), x(n,2), x(n,3), ..., x(n,M)\}$ for which $X =$ 'i'. Formally, $m_i(n) = \sum_{k=1}^{M} I(x(n,k) = 1)$ where $I(.)$ is the indicator function. Then, the values of $p_i(n), 1 \leq i \leq r$, are updated in the following way:

$$p_i(n+1) \leftarrow \frac{m_i(n)}{M}p_i(n) + \lambda(p_i(n) - \frac{m_i(n)}{M}). \quad (1)$$

The reader should note that the above algorithm is a generalization of Oommen and Rueda's original SLWE algorithm [13]. In fact, when $M = 1$, the above updated equation coincides with the original algorithm devised in [13].

The properties of the estimator are catalogued and proven below.

THEOREM 1. *Let the parameter S of the multinomial distribution be estimated by $P(n)$ at time 'n' as per equation (1). Then, $E[P(\infty)] = S$.*

Proof.

The proof is omitted here and can be found in an extended version of the current article [14].

The next result deals with the rate of convergence of the mean of the estimator.

THEOREM 2. *The rate of convergence of P is* fully *determined by λ.*

Proof.

The proof follows directly from the corresponding proof in [13]. It is omitted to avoid repetition.

4.2 Theoretical Results: Triggering Rule Reordering

For triggering the decision to attempt RR in a dynamic environment, we will use two types of approaches: Schedule-based rule ordering and Performance-triggered rule ordering. In simple terms, the Schedule-based RR will re-order the rule after a fixed number of packets have been received. On the other hand, the Performance-triggered RR will reorder the rules whenever the performance of the current policy degrades. Obviously, the problem with Schedule-based approaches is that of determining the periodicity of change. While changing the rule ordering too frequently results in unnecessary computation, if it is changed with too low a frequency, it results in a system that is unable to track the environments. As opposed to this, Performance-triggered ordering can avoid both these trends if a degradation can be detected. However the efficiency of such a scheme is dependent on how fast the degradation can be detected, which is actually quite related to resolving the change detection problem.

These two forms of mechanisms for triggering the RR, i.e., either periodically or Performance-triggered, are described in detail in the experimental results, Section 5.

With regard to Algorithm 2 we now prove a central result related to the condition that we use for swapping two rules,

namely Δ_{new}. We will show that Δ_{new} is simply the difference between the average matching time before and after swapping. Indeed, we will prove two important properties of the RR algorithm which are the following:

- Whenever a swapping is performed, the average matching time of the firewall is decreased;
- The swapping condition based on the concept of the swapping window will preserve the integrity of the firewall.

In what follows, we shall use the notation that for any rule r_i, located at position $r_i.pos$, the associated probability of it being invoked is $r_y.prob$.

4.2.1 The Swapping Condition

THEOREM 3. *The difference of the average matching time before and after swapping two rules r_x and r_y is given by: $\Delta_{new} = (r_y.prob - r_x.prob).(r_y.pos - r_x.pos)$*

Proof.

The proof can be found in [14].

4.2.2 Preserving Policy Integrity: Consistent Rule Re-ordering

Due to space limitations, we will merely present the theorems in this section without proofs. The proofs of the corresponding theorems can be found in an extended version of the current article [14].

THEOREM 4. *A rule r_k does not introduce inconsistency (i.e., it obeys all precedences relationships) if:*

$$preceding\_min(r_k) < r_k.pos < suceeding\_max(r_k).$$

THEOREM 5. *If $r_y.pos < succeeding\_max(rx)$* **AND** *$r_x.pos > preceding\_min(r_y)$*, then swapping r_x and r_y will not introduce inconsistency.

THEOREM 6. *Suppose that: $r_x.pos < succeeding\_max(r_y)$* **AND** *$r_y.pos > preceding\_min(r_x)$* then swapping r_x and r_y will not introduce inconsistency. □

5. EXPERIMENTAL RESULTS

In this section we will describe the experimental results obtained by testing our algorithm on a rigorous suite of environments. The experiments were divided into two categories, those involving **Static** and **Dynamic** environments respectively. While the static experiments were designed in such a manner that they were capable of only verifying the RR algorithm, the dynamic experiments verified the overall firewall optimizer. All together, we conducted six experiments, namely three static and three dynamic experiments.

5.1 Performance Metric: The Average Matching Time

The authors of [15] defined a metric describing the average matching time of an Access Control List (ACL). This metric can be applied to a firewall precisely because a firewall policy is comprised of ACL rules with dependency relationships. The following describes how the metric is calculated.

Let θ_i represent the matching probability of a rule r_i in R. Then the average matching time of the rule is:

$$r_i * \theta_i$$

In other words to find the average matching time, we have to simply multiply the rule r_i's probability with its current position in the firewall. Extending the above to the firewall, R, the average matching time of the firewall R can be denoted as,

$$\sum_{i=1}^{N} r_i * \theta_i, \text{ for } N > 1.$$

Thus, the average matching time is defined as the average number of rules that a packet must be compared against before a match is found. For example, if a policy R has an average matching time of 2.6, it means that on average, 2.6 packets will be compared against the rules, $\{r_i\}$, in R before a match is found. From this, it is apparent that to optimize a firewall, the average matching time of the firewall must be low. By a simple analysis one sees that this can be achieved by ensuring that the rules with high probabilities are at the top of the firewall.

5.2 Experimental Environment

The experiments were conducted on virtual machine instances created on the *Alto Cloud* cloud service at the *Oslo and Akershus University College of Applied Sciences*. All the instances were obtained using an *ubuntu 14.14* server image provided in the cloud.

In order to test the algorithms and the resulting firewalls, we needed two machines. Machine1 (M1) would run the firewall and the optimization algorithms. Machine2 (M2) would generate network traffic using a traffic generating script. However, because the firewalls being tested contained rules with random source and destination IP addresses, the traffic generating script could not send the traffic through the internet because it would have been lost and never reached the fire-

wall at M1. This was because there were no hosts in the environment that possessed those IP addresses. Consequently, in order to solve the problem, we needed a direct connection between M1 and M2. This connection was created by changing M2's default gateway to the IP of M1 so that all traffic from M2 was routed through M1. This ensured that the spoofed IPs in the network traffic generated by the traffic generating script running on M2, would reach the firewall at M1. Figure 2 illustrates this.

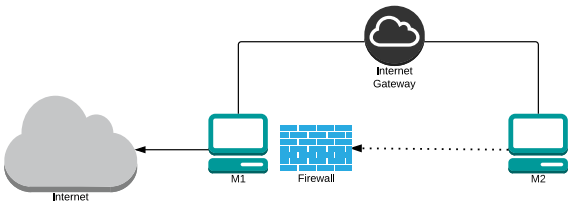

Figure 2: Proposed firewall testing environment.

5.3 Schedule-based Rule Re-ordering with Dynamic Traffic

The intention of this experiment was to test both the RR algorithm and the Batch-oriented Weak Estimator algorithm in a dynamic network using a Schedule-based re-ordering policy. The schedule policy was based on the quantity of packet matches in the firewall.

The experiment used two Zipf distributions based on the firewall in Table 2.

The first distribution, **Zipf_dist_X**, gave a higher probability to the rules in the group {E - H}, while the second distribution, **Zipf_dist_Y**, gave a higher probability to the rules in the group {A - D}. The firewall optimizer script ran the RR algorithm for every 100 packet matches generated by the traffic generating script using the Zipf_dist_X distribution. After 1,000 packets had been matched, the traffic generator would switch the distribution to Zipf_dist_Y, while the optimizer script would continue to attempt RR every 100 packet matches.

With regard to the metric of comparison, for every iteration of the firewall optimizer, we calculated the average matching time of the current firewall policy configuration, using both the current Zipf distribution probabilities and the probability values estimated by using the Batch-oriented Weak Estimator algorithm. Such a process was able to produce the true average matching time and the estimated matching time per packet matched. A base line average matching time was also calculated, which was simply the the average matching time of the firewall without any RR. Storing these values as tuples, where each was stored with the current number of packet matches at the time of calculation, enabled us to create a graph displaying the improvement rate of the average matching time for the performance of the firewall optimizer script.

The X-axis of the graph represents the total number of packet matches and the Y-axis represents the average matching time. On this graph, we plotted the progression of the three different matching time metrics.

5.3.1 *Results Obtained*

As mentioned earlier, the intention of this experiment was to test both the rule order and traffic aware algorithms using a schedule based re-ordering policy in a dynamic network. The resulting graphs for this experiment are given in Figures 3 and 4 respectively.

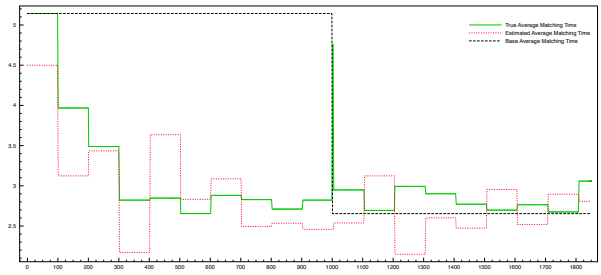

Figure 3: The results obtained from the first data set comparing our algorithm with a traditional schedule-based re-ordering policy.

The results of the first experiment demonstrate that the algorithms behaved as expected. We observe that both the **True** and **Estimated** average matching times start with high values, representative of a poorly-optimized rule ordering. They both, thereafter, start to gradually improve their times. However, there are some fluctuations in the results that leads to a spiking behavior. These spikes might be because of the nature of the traffic generator, because it does not consistently guarantee that packets with high probabilities are always chosen. Rather, the generator uses a "roulette wheel" function in order to decide which rule is to be tested. Thus, we might end up with rules with relatively low associated probability being chosen at random and being sent to the firewall, causing the observed fluctuations.

When comparing the **True** and **Estimated** average matching times, we observe that they both match, relatively closely, with the **Estimated** values being consistently slightly below the true average matching times. Besides these observations,

			Source		Destination			
No.	Unique Name	Proto.	IP	PORT	IP	PORT	Action	Prob.
1	A	UDP	190.1.*	*	*	90	accept	0.1147
2	B	UDP	190.1.1.*	*	*	90-94	deny	0.0812
3	C	UDP	190.1.2.*	*	*	*	deny	**0.4286**
4	D	UDP	190.1.1.2	*	*	94	accept	**0.1866**
5	E	TCP	190.1.*	*	*	90	accept	0.0621
6	F	TCP	190.1.1.*	*	*	88	deny	0.0499
7	G	TCP	190.1.1.2	*	*	88-94	deny	0.0415
8	H	TCP	190.1.2.*	*	*	*	accept	0.0353

Table 2: The firewall policy used for the experiment.

the base line time behaves as expected: it starts with a high average packet matching time until the switch, at which point it decreases rather sharply, and attains a matching time that is relatively close to the optimal.

The results of the second experiment are given in Figure 4.

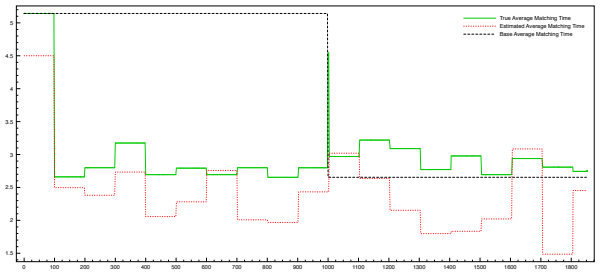

Figure 4: The results obtained from the second data set comparing our algorithm with a traditional schedule-based re-ordering policy.

These graphs display more unexpected results as there seems to have been more fluctuations. While the base line times are as expected, the **True** and **Estimated** times seem to be too flat. Again this could simply be due to the random phenomena due to the traffic generator. More importantly, we also observe that the **True** and **Estimated** times are not relatively aligned anymore, which might be because of the generally low updates to each rule as it matches a packet given by the weak estimator function.

5.4 Performance-Triggered Rule Re-ordering using a Sliding Window

The intention of this experiment was to observe the behavior of the algorithms when using a Performance-triggered criterion. Such a Performance-triggered criterion was based on a sliding window comprising of the most recent values of the estimated average matching time of the firewall. The experiment consisted of two parts both of which used the same Zipf distribution throughout the experiment. However, the first part shuffled the Zipf distribution at the traffic generator after every 500 packets sent. The second part shuffled the distribution after every 1,000 packets sent.

The firewall optimizer script ran the RR algorithm according to a Performance-triggered condition. The condition consisted of a list of the latest average matching times of the firewall. With each new calculation of the average matching time, the value was added to the list and if the list was full, the oldest element would be removed in order to make space for the latest value. This is, essentially, a sliding window. In order to decide whether to run RR or not, the optimizer script determined the trend of the sliding window. If the trend demonstrated that the average matching time was increasing, the RR procedure would be invoked. Otherwise, the RR procedure would not be called. The trend was calculated by computing the average of all but the latest value in the sliding window, and the average was compared against the latest value. If the average was greater, RR would run; otherwise, it would not.

The results enabled us to create a graph, in which the X-axis represented the total number of packet matches, and the Y-axis represented the average matching time.

5.4.1 *Results Obtained*

The results obtained by running this experiments are given in Figures 5 and 6.

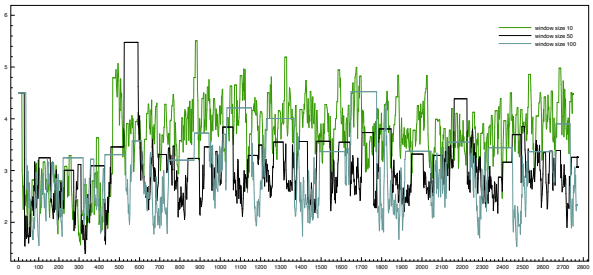

Figure 5: The graph obtained for the experiment with a dynamic environment where the distribution switched every 500 packets sent.

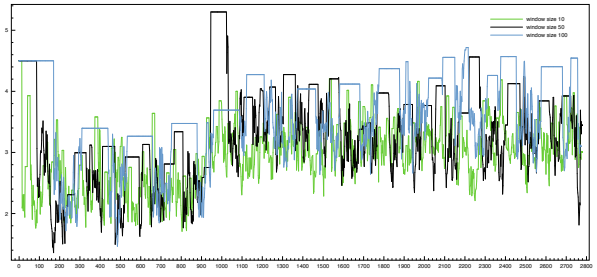

Figure 6: The graph obtained for the experiment with a dynamic environment where the distribution switched every 1,000 packets sent.

From observing the results we see that, in general, the average matching time decreased with the window size. The reason for this is because the scheme is provided with more information to determine if the trend displays an overall increase or decrease in the matching time. A small window size will, generally speaking, yield a lot of false positives resulting in a RR that occurs too early. For example, we notice that in Figure 5, when the window size is 10, the average matching time is consistently higher than for a window size equaling 50 and 100. However, in the graph where the distribution was switched after 1,000 (rather than just 500) packets, we observe that even a window size of 10 is able get comparatively good results relative to window sizes of 50 and 100. The reason for this is that the network traffic state will stay in a relatively stable state for a longer time span until the switch occurs. Of course, there are some fluctuations here, but they can be be caused by the random nature of the traffic generator.

Overall, the results match the expected results.

6. CONCLUSION

The main goal of this paper was to investigate how we could optimize a firewall's rule ordering using the network's traffic statistics.

The problem statement was addressed by developing two algorithms to achieve the Rule Re-ordering (RR) in order to optimize the firewall's rules in a dynamic network. The first algorithm was a RR algorithm. It was distantly based on the philosophy introduced in [1]. However, our algorithm used more complex criteria for initiating RR, and we experimentally demonstrated that it was able to reduce the average matching time by as much as 68% than the algorithm due to [1]. Our second main contribution was to devise a traffic-aware algorithm. It was based on the weak estimator algorithm proposed in [13]. However, it was modified to accommodate a batch updating of the rule probabilities rather than having to rely on keeping track of every packet in the system in order to update the rule probabilities.

Through various rigorous experiments, we have been able to show that the firewall performance optimizer worked very well, and that it was able to re-order the rules by using dynamic and time-varying information gleaned from the network.

References

[1] E. W. Fulp, "Optimization of network firewall policies using ordered sets and directed acyclical graphs," in *Proc. of IEEE Internet Management Conference*, 2005.

[2] L. Rueda and B. J. Oommen, "Stochastic automata-based estimators for adaptively compressing files with nonstationary distributions," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 36, no. 5, pp. 1196–1200, 2006.

[3] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," *CISCO white paper*, vol. 1, 2011.

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

[5] S. Acharya, J. Wang, Z. Ge, T. Znati, and A. Greenberg, "Traffic-aware firewall optimization strategies," in *Communications, 2006. ICC '06. IEEE International Conference on*, vol. 5, June 2006, pp. 2225–2230.

[6] C. Benecke, "A parallel packet screen for high speed networks," in *Computer Security Applications Conference, 1999.(ACSAC'99) Proceedings. 15th Annual*. IEEE, 1999, pp. 67–74.

[7] O. Paul, M. Laurent, and S. Gombault, "A full bandwidth atm firewall," in *Computer Security-ESORICS 2000*. Springer, 2000, pp. 206–221.

[8] L.-P. Bigras, M. Gamache, and G. Savard, "The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times," *Discrete Optimization*, vol. 5, no. 4, pp. 685–699, 2008.

[9] V. Grout and J. McGinn, "Optimisation of policy-based internet routing using access control lists," in *Proceedings of the 9th IFIP/IEEE Symposium on Integrated Network Management*, 2005.

[10] A. Schrijver, "On the history of combinatorial optimization (till 1960)," *Handbooks in Operations Research and Management Science: Discrete Optimization*, vol. 12, p. 1, 2005.

[11] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of discrete mathematics*, vol. 5, pp. 287–326, 1979.

[12] A. Tapdiya and E. Fulp, "Towards optimal firewall rule ordering utilizing directed acyclical graphs," in *Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on*, Aug 2009, pp. 1–6.

[13] B. J. Oommen and L. Rueda, "Stochastic learning-based weak estimation of multinomial random variables and its applications to pattern recognition in non-stationary environments," *Pattern Recognition*, vol. 39, no. 3, pp. 328–341, 2006.

[14] R. Mohan, A. Yazidi, B. Feng, and B. J. Oommen, "On optimizing firewall performance in dynamic networks by invoking a novel *swapping window*-based paradigm," *Unabridged version of this paper. To be Submitted for Publication.*

[15] V. Grout, J. Davies, and J. McGinn, "An argument for simple embedded acl optimisation," *Computer Communications*, vol. 30, no. 2, pp. 280–287, 2007.