Towards Optimal Firewall Rule Ordering Utilizing Directed Acyclical Graphs

Ashish Tapdiya and Errin W. Fulp
Department of Computer Science
Wake Forest University
Winston Salem, NC, USA
nsg.cs.wfu.edu
Email: {*tapda7* | *fulp*}*@wfu.edu*

***Abstract*—Firewalls enforce a security policy by inspecting packets arriving or departing a network. This is often accomplished by sequentially comparing the policy rules with the header of an arriving packet until the first match is found. This process becomes time consuming as policies become larger and more complex. Therefore determining the appropriate action for arriving packets must be done as quickly as possible.**

The process of packet header matching can be improved if more popular rules appear earlier in the policy. Unfortunately, a simple sorting algorithm is not possible since the relative order of certain rules must be maintained in order to preserve the original policy intent. Utilizing Directed Acyclical Graphs (DAGs) to represent firewall policy, this paper will introduce a novel rule sorting technique. The technique is capable of considering sub-graphs of rules (inter-related by precedence constraints) and compare the advantage of placing and merging the rules that comprise them. Experimental results using a variety of policies will show that the proposed algorithm is able to find the optimal order in 98% of the example policies, which is substantially higher than other methods.

***Index Terms*—Security, network firewall, security policy, rule ordering**

I. Introduction

A firewall forms the first line of defense for a network and enforces a security policy by filtering malicious or unsolicited packets from incoming packet stream [1], [2]. In a list based firewall this is accomplished by sequentially comparing the incoming packet with the rules until the appropriate match is found. Once the matching rule is determined the associated action is performed on the packet [1], [2]. Hence, as the size and complexity of policy increases, incoming packets will have to be compared against a large set of rules and packet inspection will require more work [2], [3]. Moreover, suboptimal placement of rules in the policy can greatly reduce the efficiency of the firewall [3], [4].

Firewall rule re-ordering can reduce the latency experienced by the passing traffic. Unfortunately, finding the best order has been shown to be an $\mathcal{NP}$-hard problem [5]. Researchers have developed different heuristics to find better firewall rule orders. For example in [6], a simulated annealing based algorithm for firewall rule reordering was proposed. However, the upper bounds on the computational cost of the algorithm are not discussed and represents a heavyweight solution which may not be suitable for large policy sizes. In [7], Heuristic Optimal Rule Ordering (HORO) algorithm is presented. The HORO 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 sequentially selects rules based on the descending order of weights. Selected rule along with all its dependent rules are sorted and inserted in the sorted list while maintaining the integrity of the policy. In [8], Simple Rule Sorting (SRS) algorithm is introduced, which sorts independent neighboring rules based on the non-increasing probabilities.

In this paper we propose a novel heuristic sorting technique for determining firewall policy rule ordering which reduces the average number of rule comparisons while maintaining the intent of the policy. A key difference between the proposed sorting technique and previous methods is the ability to sort and merge groups of interdependent rules which can occur in actual security policies. In these cases, certain rules must appear before others in order to maintain the original intent of the policy. The performance of the proposed algorithm is compared with the HORO and SRS algorithms using randomly generated policies. The experimental results will indicate that the proposed algorithm can find the optimal rule ordering in 98% of the example cases.

The remainder of this paper is organized as follows. Section II provides with the definition of a firewall rule, firewall policy, and the Directed Acyclical Graph (DAG) model for representing firewall policy. Difficulty of the problem and optimization equation are presented in section III. Section IV presents with a novel heuristic rule sorting technique. The performance of the sorting algorithm described in section IV is investigated experimentally in section V. Finally, section VI summarizes this paper and discusses some areas for future work.

II. Firewall Policy Model

Inspecting traffic sent between networks, a firewall provides access control, auditing, and traffic control based on a security policy [1], [2]. This section explains the building blocks of network firewalls. It includes defining firewall policies, network packets, and firewall rules. In addition, this section describes rule intersection which forms the basis for transformation of a

978-1-4244-4581-3/09/$25.00 ©2009 IEEE

traditional list-based security policy into a policy DAG which is better suited for visualization and optimization.

A. Firewall Rules, Packets and Policies

A firewall rule r is an ordered set of parameters $r = (r[1], ..., r[k])$, where $k > 1$. For the Internet firewall rules are commonly represented as an ordered set of 5 parameters comprising: protocol, source IP address, source port, destination IP address, destination port [1], [2]. Each parameter $r[l] \in r$ represents a set of values. In addition to these parameters each rule r has an associated action as seen in Table I. The rule action is only performed after an incoming packet matches that particular rule.

Every incoming packet has a header and a payload. The header indicates the source and destination of the packet while the payload is the actual data. The header of an incoming packet d has the same parameters as a firewall rule. However, a firewall rule parameter value can be range or a single value whereas, a packet parameter value can only be a single value. Hence, the set of packet header parameters corresponding to each rule parameter is denoted by $(d[1], ..., d[k])$.

A firewall policy is traditionally an ordered list of n rules, denoted as R = (r_1,r_2,r_3,..., r_n), where r_i is the i^{th} rule in the policy. Given this structure many implementations represent the policy as a linked list [1].

TABLE I
EXAMPLE SECURITY POLICY FOR WHICH THE SGM ALGORITHM ORDERING IS NOT THE OPTIMAL ORDERING.

		Source		Destination			
No.	Proto.	IP	Port	IP	Port	Action	Prob.
1	UDP	190.1.*	*	*	90	accept	0.0645
2	UDP	190.1.1.*	*	*	90-94	deny	0.161
3	UDP	190.1.2.*	*	*	*	deny	0.2258
4	UDP	190.1.1.2	*	*	94	accept	0.2587
5	*	*	*	*	*	deny	0.29

A firewall policy R is considered comprehensive if for every possible packet d, a match is found in R. In order to make a security policy comprehensive, often a default rule is added to the end of the policy that matches every incoming packet and performs the deny action on it. Table I represents an example comprehensive security policy, where r_5 is the default rule.

B. Packet Matching Process

As described in [8], [9] a match occurs between a rule and a packet if each parameter value in the packet header is a proper subset of the corresponding parameter value in the rule. Let the packet d and rule r_i match be denoted as,

$$d \Rightarrow r_i \quad \text{iff} \quad \forall \ l, \quad d[l] \subseteq r_i[l], \quad l = 1, ..., k$$

Given a policy R and packet d, multiple matches may be possible. The 'match policy' describes which of the multiple matching rules will be applied. There are three commonly used matching mechanisms 'first match', 'last match' and 'best match'. In Best Match, an incoming packet d is compared against the entire policy. The rule that matches the closest to the packet header is selected and the action associated with it is performed on the packet. The closeness is measured by the narrowness of the rule's parameters, or the longest matching prefix [10]. This matching mechanism is used by the routers in routing packets. In Last Match, an incoming packet d is sequentially compared against each rule $r_i \in$ R, beginning at the first rule in policy, until the end of the policy. In First Match, an incoming packet d is sequentially compared against each rule $r_i \in$ R, beginning at the first rule in policy, until a match is found ($d \Rightarrow r_i$). The match operation ends after finding the first rule that matches the packet and the associated rule action is performed on the packet. First match is the most popular method for network firewalls. It is simple to implement, and will be the assumed matching policy for this paper.

C. Rule Intersection and Policy Model

For first matching policies, an important characteristic of firewall rules is precedence, which is determined by intersection of rules. Two rules r_i and r_j intersect with each other if their parameter values have nonempty intersection across all corresponding parameters. Intersection among pair of rules is possible because of the allowance of range representation across different parameters comprising a rule. The rule r_i and the rule r_j intersection can be denoted as,

$$r_i \Downarrow r_j \quad \text{iff} \quad \forall \ l, \quad r_i[l] \cap r_j[l] \neq \phi, \quad l = 1, ..., k$$

As described in the section II, a firewall policy R is an ordered list of rules and a packet d is sequentially compared against the rules in R beginning at the first rule until a match is found. Although described as a list, a DAG G = (R,E), can be used to model a firewall policy, where R is the set of rules in policy and E represents the set of precedence relationships between rules [8], [9]. An edge exists between rule r_i and rule r_j, if $i < j$ and rules intersect. A firewall policy can be transformed into corresponding policy DAG by iterating and comparing a rule with all successor rules using the intersection operator $\Downarrow$ [5].

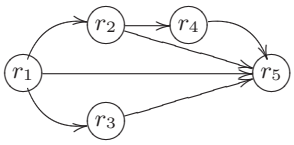
For policy rules that intersect, exchanging their relative order may change the intent of policy. For example, consider rules r_1 and r_5 ($r_1 \Downarrow r_5$) from the policy given in Table I. If rule r_1 is swapped with rule r_5 then, the outcome will be a denial of all incoming packets to this firewall. Hence the swapping of rules introduces an anomaly in the policy. However, if rule r_2 is interchanged with rule r_3, the intent of policy is maintained because $r_2 \not\Downarrow r_3$. Thus certain rules have precedence relationships between them which need to be preserved in any reordering of the rules.

Figure 1(a) represents the DAG corresponding the security policy presented in Table I. There exist an edge between the rules r_2 and r_4 because $r_2 \Downarrow r_4$. However, there does not exist an edge between the rules r_2 and r_3 because $r_2 \not\Downarrow r_3$. All linear permutations of the policy DAG seen in Figure 1(a) will maintain the integrity, this is proven in [8], [9].

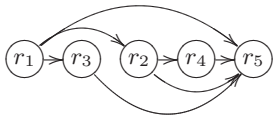
III. DIFFICULTY OF RULE ORDERING

Certain firewall rules have a higher probability of matching a packet than others. Hence, it is possible to develop a policy

profile over time that indicates frequency of rule matches. Let P = $[p_1, p_2, p_3, ..., p_n]$ be the policy profile, where p_i is the probability that a packet will match rule r_i. A rule order that requires the fewest number of rule compares on average is referred to as the optimal rule order. Optimal rule ordering can be determined by utilizing the probability associated with each rule as a comparison criteria for reordering the rules.



(a) Policy DAG.



(b) SGM and optimal order, *R(π)* = 2.886.

Fig. 1. Different policy DAGs for the firewall rules given in Table I.

Let R$'$ = [r_1',r_2',r_3',...,r_n'] be a linear arrangement of R. Then, [r_1',r_2',r_3',...,r_n'] = π[r_1,r_2,r_3,...,r_n] and [p_1',p_2',p_3',...,p_n'] = π[p_1,p_2,p_3,...,p_n]. Hence, a linear arrangement π of the rules in R is sought that minimizes equation,

$$\mathrm{R}(\pi) = \sum_{i=1}^{n} p_i' \; t_i \; .$$

where t_i is the total time to match a packet against rule r_i.

In the absence of precedence relationships, the rules are completely independent of each other and the average delay per packet is minimized by sorting the rules in non-increasing order according to the probabilities [11]. The problem of finding optimal firewall rule set ordering is a reduction from the problem of single machine job scheduling with precedence constraints [5]. Since single machine job scheduling problem with precedence conditions has been proved to be $\mathcal{NP}$-hard thus the problem of optimal firewall rule set ordering is also $\mathcal{NP}$-hard [8], [9].

IV. Sub-Graph Merging (SGM) Algorithm

The SGM algorithm is a heuristic firewall policy sorting technique. It has the ability to sort the rules in R that have precedence relationships between them (a subgraph in the DAG), while maintaining the integrity of the input firewall policy. The complete SGM algorithm is given in Figure 3, and as seen in the figure, certain variables and data structures are required to support its operation. Let G(r_i) be the set of rules in the subgraph of rule r_i, including r_i. For example in Figure 1 G(r_4) = $\{r_1, r_2, r_4\}$. Let G*(r_i) be the set of rules in the subgraph of rule r_i excluding itself, therefore in Figure 1 G*(r_4) = $\{r_1, r_2\}$. Now, let Z(r_i) be the sum of probability of all the rules in set G(r_i). Therefore,

$$\mathrm{Z}(r_i) \;\; = \sum_{\forall \; k \; \in \; \mathrm{G}(r_i)} p_k,$$

where p_k is the probability of rule *k*. For example Z(r_4) = p_1 + p_2 + p_4 = 0.0645 + 0.161 + 0.2587 = 0.4842. Also, let W(r_i) be the cardinality of set G(r_i). Hence, Z(r_i) / W(r_i) is the average probability of the subgraph G(r_i).

Let S be the sorted policy represented by a FIFO Queue and, initially S = ϕ. Let Q be the list of rules to be sorted and initially Q = R. Let X be the vector containing Z(r_i), for all r_i ∈ R. Also, let C be the vector containing W(r_i), for all r_i ∈ R. Z(r_i) and W(r_i) for all r_i ∈ R, can be precomputed in $\mathcal{O}(n^2)$ and stored in X and C respectively. Let PROB be the vector containing the probabilities for all r_i ∈ R, DEP represent the dependency matrix for the policy DAG in question.

During each pass, the algorithm selects the rule with the highest average subgraph probability from the graph of rules available during that pass (algorithm lines 5 - 9). Recall the average subgraph probability for any rule is the average of probability of rules present in its subgraph. The selected rule is then inserted in the list of sorted rules, S, if it has no rules dependent on it (algorithm lines 14 - 18). Otherwise, the algorithm iteratively sorts the subgraph of its dependents until it finds a rule that has no dependent rules (algorithm lines 20 - 37) and inserts that rule in the list of sorted rules. The algorithm then updates any data structures (algorithm lines 39 - 45) and repeats the process (return to line 2 of the algorithm) until all rules have been placed in S.

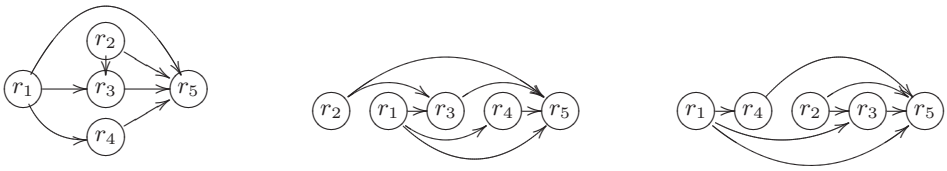
An important attribute of any firewall rule sorting technique is the ability to maintain rule integrity. Once the algorithm has reordered the policy, the new rule order must maintain the original intent. A proof that the SGM algorithm maintains integrity follows.

Theorem 3.1 *Given a first match policy, the rule ordering produced by the* SGM *algorithm will always maintain the policy integrity.*

Proof Assume a policy DAG G is constructed from security policy R. Let r_i be the *selected* rule. If G*(r_i) = ϕ then algorithm SGM can insert rule r_i in the queue of sorted rules. Insertion of rule r_i will not affect integrity since rule r_i does not have any rule dependent on it, as indicated by G*(r_i). If G*(r_i) ≠ ϕ then, all r_k ∈ G*(r_i) are sorted until a rule r_j is not found for which G*(r_j) = ϕ; therefore, integrity is maintained.

A. Example That Results In Non-Optimal Ordering

Although the objective is to find the optimal rule ordering, it is possible that SGM will not. This section provides such an example. For the DAG in Figure 2, SGM ordering is (r_2,r_1,r_3,r_4,r_5) where as the optimal ordering is (r_1,r_4,r_2,r_3,r_5). In the first pass inside the outer while loop, rule r_5 has the highest average subgraph probability of 0.2. However, since it has dependent rules, its subgraph excluding itself is sorted iteratively. Hence, in the first pass inside the inner while loop, rules r_4, r_3, r_2 and, r_1 contend for selection. Rule r_3 is selected as its average subgraph probability is highest. Now, since r_3 has rules r_2 and r_1 as dependents, hence sorting is continued inside the inner while loop. In the next pass, r_2 gets selected over r_1 and since it has no dependents, is inserted in S. Thus, SGM algorithm selects r_2 in the first pass where as optimal ordering has r_1 in the first place.



(a) Policy DAG. (b) Original order, $R(\pi)$ = 4.3. (c) Sorted order, $R(\pi)$ = 3.8.

Fig. 2. Different policy DAG representations of the firewall rules given in Table II.

SGM algorithm makes a mistake during the pass inside the inner while loop in which rules r_2 and r_1 contend. Rule r_2 has higher average subgraph probability than r_1 and hence gets selected. This pushes rule r_4 down by at least one place because r_4 has r_1 as a dependent, which did not get removed in the current pass. However, if r_1 is selected ahead of r_2 when rules r_1 and r_2 contend, then it allows rule r_4 to get selected in the next pass. As r_4 has high probability, its selection in the second pass as compared to SGM algorithm's fourth pass, results in a smaller average delay.

B. Complexity Analysis

This section describes the worst case time complexity for SGM algorithm presented in Figure 3. The worst case occurs when the policy DAG is completely connected and the average subgraph probability of rules are in increasing order such that $(\text{X}[r_i]/\text{C}[r_i]) < (\text{X}[r_j]/\text{C}[r_j])$ where, $i < j$ for $i, j \in [1, n]$. The numbers under and over the braces in Equation 1 correspond to the line numbers presented in Figure 3.

$$\overbrace{\sum_{i=1}^{n}}^{2-46}\Bigg(\underbrace{\sum_{k=n-i+1}^{1}}_{5-9}\overbrace{c_1}^{6-8}+\overbrace{\underbrace{\sum_{k=n-i+1}^{1}}_{14-18}\Big(c_2+c_3\Big)}^{12-37}+\underbrace{\sum_{j=k-1}^{1}}_{22-36}\overbrace{c_4}^{23-33}\Bigg)+\underbrace{\sum_{k=i-1}^{0}}_{39-45}\overbrace{c_5}^{40-44}\Bigg) . \tag{1}$$

$$\mathcal{O}(n^3) .$$

V. Experimental Results and Analysis

This section compares the performance of the proposed approach with the HORO and SRS algorithms. Recall the HORO algorithm takes as input an optimization limit that represents an upper bound on the total weight of the selected rules in the optimization process. In our implementation of the HORO algorithm the sum of weight of all rules in the policy was used as the optimization limit. Therefore, every rule in the firewall was considered an active rule. Experiments were done using simulation with randomly generated policies. A firewall policy can be characterized by three parameters: the policy size n, the number of edges e, and the policy profile P. For simulations, these three parameters are varied to represent a wide range of policies. For example, policy profiles were selected randomly, where any feasible profile was equally likely. Similarly, for a given number of edges, their placement was randomly selected.

```
1  function policySort(S, Q, X, C, PROB, DEP)
2  while(Q ≠ φ)
3      /* Select the best rule in Q */
4      set r_b to a rule in Q
5      for(∀ r_j ∈ Q and r_j ≠ r_b)
6          if((X[r_b]/C[r_b]) < (X[r_j]/C[r_j])) then
7              set r_b to r_j
8          end
9      end
10
11     bool selected = false
12     while( !selected )
13       /* Check if the selected rule has any dependents */
14       if(C[r_b] == 1) then
15           add r_b to S and remove r_b from Q
16           set selected to true
17           set r_selected to r_b
18       end
19       else
20           /* Select the best rule from G*(r_b) */
21           bool temp = false
22           for (i = 1; i < b; i++ )
23               if(DEP[r_i][r_b] == 1) then
24                   if( temp == false ) then
25                       set r_tmp to r_i
26                       set temp to true
27                   end
28                   else
29                       if((X[r_tmp]/C[r_tmp]) < (X[r_i]/C[r_i]))
30                           set r_tmp to r_i
31                       end
32                   end
33               end
34           end
35           set r_b to r_tmp
36       end
37     end
38     /* Update Data Structures */
39     for (i = selected+1 ; i ≤ n ; i++)
40         if (DEP[r_selected][r_i] == 1)
41             set DEP[r_selected][r_i] to 0
42             decrement C[r_i] by 1
43             X[r_i] = X[r_i] - PROB[r_selected]
44         end
45     end
46 end
```

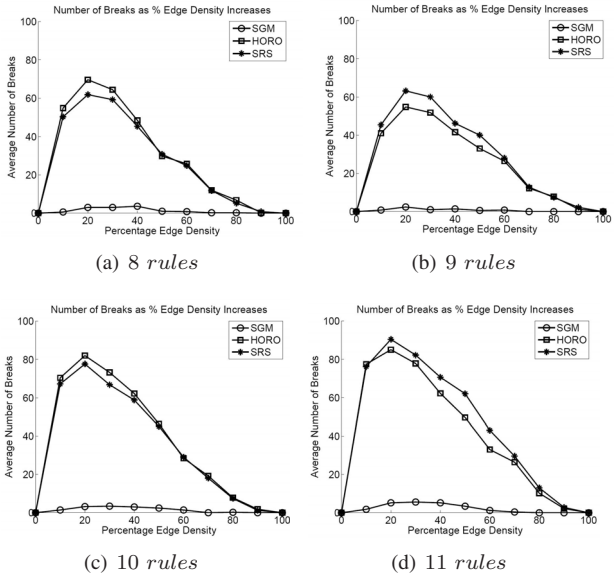
Fig. 3. SGM algorithm.

In sections V-A and V-B experiments were performed with small policy sizes. For each run the brute force algorithm was

TABLE II
EXAMPLE SECURITY POLICY FOR WHICH SGM ORDERING IS NOT OPTIMAL ORDERING.

No.	Proto.	Source IP	Source Port	Destination IP	Destination Port	Action	Prob.
1	UDP	190.1.*	*	*	90	accept	0.0585333
2	UDP	190.1.1.*	*	*	88	deny	0.0728863
3	UDP	190.1.1.2	*	*	88-94	deny	0.156762
4	UDP	190.1.2.*	*	*	*	accept	0.123346
5	*	*	*	*	*	deny	0.5884724

also executed, which enabled us to identify the cases for which SGM, HORO and SRS algorithms failed to find optimal order. Each failure case will be referred to as a "break". Given the difficulty of the problem, determining the optimal ordering for large policies is infeasible; hence, in section V-C for each run only the average number of rule comparisons was used to evaluate performance.



(a) 8 $rules$ (b) 9 $rules$

(c) 10 $rules$ (d) 11 $rules$

Fig. 4. Impact on average number of breaks as percentage edge density increases for different policy sizes.

A. Edge Density Versus Number of Breaks

This experiment compares average number of break cases for SGM, HORO and SRS algorithms, as the number of precedence edges increases. The edge density, which is the percentage of total number of possible edges, was varied between 0% and 100% in the increments of 10%. A 0% edge density indicates that the DAG is completely disconnected while a 100% edge density refers to a completely connected DAG. Also, selected policy profile remains constant throughout the experiment. For each number of edges 100 runs were performed and the number of breaks were recorded. This was repeated for policies with 8, 9, 10, and 11 rules. For each integer $n \in [8, 11]$, 5 experiments were performed.

The results presented in the graphs in Figure 4 indicate that SGM algorithm performs better than SRS algorithm. This performance increase is because SGM algorithm is capable of sorting interdependent as well as independent rules whereas, SRS algorithm can only sort independent rules. Also, SGM algorithm performs better than the HORO algorithm since during any pass SGM algorithm sorts all the rules in the DAG to select the best rule to be inserted into the sorted queue whereas HORO algorithm sorts and inserts the subgraph of the selected rule. Hence, the SGM algorithm has more possible orderings to select from.

B. Break Case Comparison

In this experiment the overlap of the break cases are investigated to help determine when one algorithm has difficulty finding optimal ordering as compared to the others. The edge density was varied between 0% and 100% in the increments of 10%. For each number of edges 100 runs were performed. The number of rules n was varied in the range from 8 to 11. The graphs in Figure 5 represents the cases for which SGM, HORO and, SRS ordering was not the same as the brute force ordering. The x axis represents the number of rules for the break case; the y axis represents the edge density for the break case; and the z axis represents the standard deviation of policy profile probability distribution for each break case. The graphs in Figure 5 enable us to visualize the algorithmic overlap for each break case.

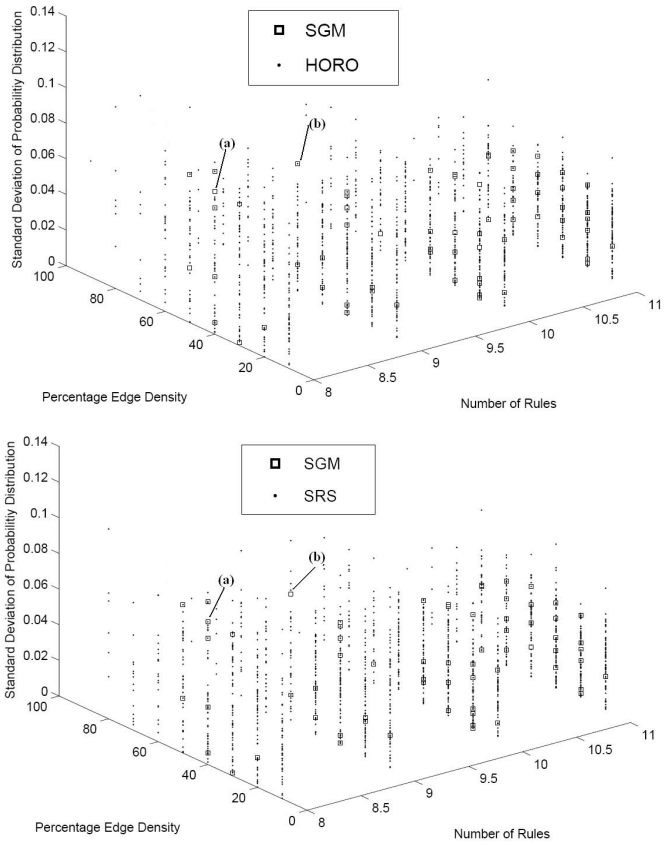


Fig. 5. Break cases as number of edges, number of rules and standard deviation of policy profile probability distribution are varied per experiment.

In Figure 5 point (a) represents the case in which SGM and SRS algorithms fail to find the optimal ordering whereas HORO algorithm ordering is same as the optimal ordering. Similarly, point (b) represents the case in which SGM and HORO algorithms fail to find the optimal ordering whereas SRS algorithm succeeds to find optimal ordering. Hence, the simulation results indicate that the SGM algorithm, on an average, breaks fewer times as compared to the HORO and SRS algorithm; however, there exists cases for which SGM algorithm can fail but HORO or SRS algorithm can find optimal ordering.

C. Percentage Improvement versus Edge Density

This experiment presents the variation in average number of rule comparisons as the edge density increases. The number of edges e were varied from 0 to e_{max} such that, $e_1 = 0, e_2 = n/10$, and $e_x = 3 * e_{x-1}$, for $x = 3, ..., y$ such that $e_y \leq e_{max}$.

The number of edges are varied using a geometric function since, the difference between the minimum and the maximum number of possible edges is very large. Also, during each experiment 10 runs were performed at each point in the edge range. This experiment was performed for policy size n equal to 500, 1000, 1500 and 2000. Also, for each policy size experiment was repeated 3 times. The average number of rule comparisons at each point in the edge range per experiment was recorded. During each experiment the hit probability of each rule is selected such that it follows Zipf distribution, which represents a more specific set of policy profiles as compared to the previous experiments. Also, the policy profile remains the same throughout the experiment.

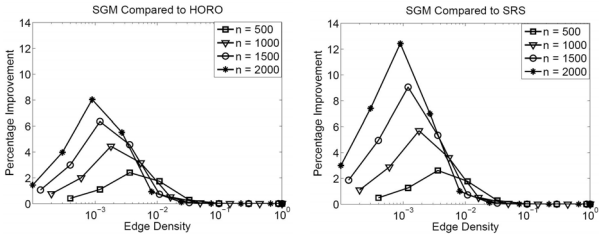


Fig. 6. SGM percentage improvement over HORO and SRS algorithms as edge density and number of rules increase.

Figure 6 shows the percentage improvement of SGM algorithm ordering's average number of rule comparisons over HORO and SRS algorithm ordering's average number of rule comparisons. It is apparent that as the policy size increases the percentage improvement of SGM algorithm increases as compared to HORO and SRS algorithms.

VI. Conclusions and Future work

Network firewalls can be improved if more popular rules appear earlier in the policy. Unfortunately, a simple sorting method is not possible, since it is proved to be a $\mathcal{NP}$-hard problem [5]. This paper presented a novel heuristic sorting technique, called the Sub-Graph Merging (SGM) algorithm, which has the capability of merging different sub-graphs of firewall rules and is statistically better than other approaches. In addition to that, this paper also compares the performance of SRS algorithm with HORO and SRS algorithms, based on the average number of cases where the algorithm is unable to find the optimal ordering and the average number of rule comparisons. Experimental results with small policy size exhibit that the proposed algorithm is able to determine the optimal order in 98% of the example policies, which is substantially higher than other methods. With large policy size, SRS algorithm's average number of rule comparisons are 8% lower than HORO and 12% lower than SRS algorithms respectively.

While the proposed algorithm has shown great promise, more research is needed to determine how frequently should the policy be sorted using SGM algorithm. Another area of research would investigate the merging and splitting of rules resulting in smaller ruleset and ruleset with independent rules respectively. Finally, there is a need to develop a centralized database with realistic firewall policies which will enable more accurate performance analysis and comparison of different firewall rule re-ordering algorithms.

Acknowledgment

The authors would like to thank Wake Forest University and GreatWall Systems, Inc. for their support. This research was funded by GreatWall Systems, Inc. via United States Department of Energy STTR grant DE-FG02-06ER86274. The views and conclusions contained herein are those of the author(s) and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the DOE or the U.S. Government.

References

[1] R. L. Ziegler, *Linux Firewalls.* New Riders, second edition, 2002.

[2] E. D. Zwicky, S. Cooper, and D. B. Chapman, *Building Internet Firewalls.* O'Reilly, 2000.

[3] C. Benecke, "A parallel packet screen for high speed networks." in *Proceedings of the* 15^{th} *Annual Computer Security Applications Conference*, 1999.

[4] S. Goddard, R. Kieckhafer, and Y. Zhang, "An unavailability analysis of firewall sandwich configuration." in *Proceedings of* 6^{th} *IEEE Symposium on High Assurance Systems Engineering*, 2001.

[5] A. Tapdiya, "Firewall policy optimization and management," Master's thesis, Wake Forest University, Computer Science Department, 2008, http://hdl.handle.net/10339/37458.

[6] E.-S. M. El-Alfy and S. Z. Selim, "On optimal firewall rule ordering," in *Proceedings of IEEE International Conference on Computer Systems and Applications*, 2007.

[7] H. Hamed and E. Al-Shaer, "On autonomic optimization of firewall policy organization," *Journal of High Speed Networks, Special issue on Security Policy Management*, vol. 13, no. 3, pp. 209–227, August 2006.

[8] E. W. Fulp, "Optimization of network firewalls policies using directed acyclical graphs." in *Proceedings of IEEE Internet Management Conference (IM'05)*, 2005.

[9] ——, "Optimization of network firewall policies using ordered sets and directed acyclical graphs." Wake Forest University, Computer Science Department, Tech. Rep., 2004.

[10] V. P. Ranganath and D. Andresen, "A set-based approach to packet classification." in *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, 2003, pp. 889 – 894.

[11] W. E. Smith, "Various optimizers for single stage production." *Naval Res. Logist. Quart.*, vol. 3, pp. 59–66, 1956.