Optimising Rule Order for a Packet Filtering Firewall

Ian Mothersole and Martin J. Reed
School of Computer Science and Electronic Engineering
University of Essex
Wivenhoe Park, Colchester
Essex, CO4 3SQ
United Kingdom
Email: {imothe, mjreed}@essex.ac.uk

***Abstract*—A heuristic approximation algorithm that can optimise the order of firewall rules to minimise packet matching is presented. It has been noted that firewall operators tend to make use of the fact that some firewall rules match most of the traffic, and conversely that others match little of the traffic. Consequently, ordering the rules such that the highest matched rules are as high in the table as possible reduces the processing load in the firewall. Due to dependencies between rules in the rule set this problem, optimising the cost of the packet matching process, has been shown to be *NP*-hard. This paper proposes an algorithm that is designed to give good performance in terms of minimising the packet matching cost of the firewall. The performance of the algorithm is related to complexity of the firewall rule set and is compared to an alternative algorithm demonstrating that the algorithm here has improved the packet matching cost in all cases.**

I. Introduction

Firewalls, in general, are devices that filter traffic between two networks - usually a protected internal network and a less trustworthy external network, like the Internet for example. Firewalls are a common computer security defence mechanism and they are used, or at least they should be used, on any computer, or network of computers, attached to the Internet. The main purpose of a firewall is to enforce a network security policy. The ways in which firewalls operate varies extensively, and can range from a layer 3 and 4 packet filter, to gateway proxy services and layer 7 deep packet inspectors. This paper will focus on layer 3 and 4 list-based packet filtering firewalls.

To ensure that network traffic is monitored effectively, the packet filtering firewall is placed at a key point in the network - usually at the gateway which links the protected network to a less trustworthy network (like the Internet). The positioning of the firewall at the network edge is often necessary because that is the best place to ensure that all traffic entering or exiting the network is monitored and filtered appropriately. Positioning a device such that all the inbound and outbound network traffic is flowing through it is always a weakness since it leads to a bottleneck effect and also introduces a single point of failure. Due to the positioning of the firewall in a network topology it is clear that as a network increases in size and/or traffic volume increases, the load put on the firewall also increases.

A list-based packet filtering firewall implements a security policy using a series of ordered rules which network packets are sequentially matched against as they pass through the device. The rule sets in firewalls can become large when there is a combination of: complex user requirements, diverse networked applications, and a need to combat increasingly sophisticated network based attacks. The packet matching provided by these rule sets is more complex than that performed by a routing table look up process as the rules perform a search over many fields in the packet and may also need to record some state information. A large rule set can have a detrimental effect on the performance of the firewall or require more expensive hardware that can process the large number of packet matches. Consequently, there is a motivation for efficient packet matching.

General efficient packet matching techniques exist, a review of some is presented by Gupta and McKeown [1]. However, these techniques do not make use of information about the relative frequency of the packets that are to be matched. The technique presented here makes use of the simple observation that the order of the rules in a rule set is important because as packets arrive at a firewall they are checked sequentially against the rules until the first match is found. If most of the packets match rules at the beginning of the list then fewer packet matching operations are required and the efficiency of the firewall is increased. The task is then to order the rules such that for a given traffic distribution, in the given matched fields, the number of packet-rule matches is minimised. In this paper we present an algorithm that orders the rules in a firewall rule set to best suit the trends in the network traffic (as given by a recent network trace file) and therefore reduce the potential number of packet-rule matches.

The rest of this paper is organised as follows: Section II presents work related to the problem including examples of rules and descriptions of rule dependencies; Section III demonstrates typical trends in network traffic giving rise to the general optimisation technique; Section IV formally introduces the problem, proposed heuristic solution and tests performed on synthetic rule sets demonstrating its efficacy; finally conclusions are drawn.

978-1-4577-0737-7/11/$26.00 ©2011 IEEE

TABLE I
A SAMPLE PACKET FILTERING FIREWALL RULE SET

	proto	src address	port	dst address	port	action
R_0	TCP	0.0.0.0/0	any	10.0.42.2	53	allow
R_1	UDP	0.0.0.0/0	any	10.0.42.2	53	allow
R_2	TCP	192.168.60.4	any	10.0.95.217	80,443	allow
R_3	TCP	192.168.60.0/24	any	0.0.0.0/0	80,443	block
R_4	TCP	0.0.0.0/0	any	10.0.95.217	80,443	allow
R_5	TCP	0.0.0.0/0	any	0.0.0.0/0	23	block
R_6	TCP	0.0.0.0/0	23	0.0.0.0/0	any	block
R_7	TCP	192.168.16.0/24	any	10.0.0.0/16	20,21	block
R_8	TCP	0.0.0.0/0	any	10.0.0.0/16	20,21	allow
R_9	TCP	192.168.0.0/16	any	0.0.0.0/0	143,993	allow

II. BACKGROUND AND RELATED WORK

A packet filtering firewall uses an ordered list of filtering rules $(R_0, R_1, \ldots, R_n)$; Table I shows an example of a firewall rule set. Network packets are checked against the rules sequentially from top to bottom until a match is found. IP network packets usually contain distinct fields from the layer 3 and layer 4 packet headers that can be used to uniquely describe the packets P belonging to a flow using the 5-tuple P={transport layer protocol, source IP address, transport layer source port, destination IP address, transport layer destination port}. This can also be extended to include different IP versions and esoteric transport layers but we will use this description without loss of generality. The elements of P may be expressed exactly, or using wild cards, to express a group of packet flows as a superset of smaller flow descriptors. Each filtering rule R is an action associated with a description of a set of packets where the action is *allow* or *block*, R={P, action = allow *or* block}. For example: R={TCP, 10.0.0.0/16, any, 0.0.0.0/0, 80, allow} allows any TCP traffic coming from the network 10.0.0.0/16, to access port 80 (HTTP) on any other network. Note that here, ranges of addresses are displayed using the common prefix/mask notation. The firewall will also have a default rule at the end (usually block) so that if a packet is not matched then as a last resort it will be automatically dropped. Rules can also have other actions like, for example, log and reject. A log action can make a record of a particular type of packet - maybe a potential attack or connection attempt on a specific port. Reject actions are similar to block but will also inform the sender that the packet was blocked.

A. Rule Dependencies

Firewall filtering rules may not be disjoint, which means that a packet could potentially match more than one rule. In this case the rules are said to be *related*. All of the possible rule relationships have been classified by Al-Shaer and Hamed [2]. These relationships may be classed as *anomalies* and in cases where the anomaly does not comply with the required policy, the relationship is said to be a *conflict*. For example, if one rule would block a packet whereas another would allow a packet then there may be a conflict with the security policy. In this work we assume that all of the relationships in a given rule set are legitimate and that no conflicts exist; if a conflict were to exist then it will be treated as any other relationship and the optimised rule set will still contain the same conflict as the original rule set. The detection of conflicts in firewall rule sets is beyond the scope of this work, however some examples of such work are given later.

Two rules may be related, but the order of the rules is only important if the actions of the two rules differ. Here we use the term *dependent* to refer to two rules that have a relationship and where the order needs to be maintained to comply with the security policy and avoid a conflict. For two or more dependent rules the rule which comes first in the rule set is said to have *higher precedence* over the subsequent related rule(s). For example Rules R_2, R_3 and R_4, from Table I are dependent because changing the order of them would alter the overall effect of the firewall policy. R_2 must precede R_3, and also R_3 must precede R_4. Rules R_7 and R_8 also display a dependency relation. Typically a dependency will occur when the policy demands either of the following: block all hosts of a network using a particular port or ports, but first allow one particular host or sub-network through; or allow all hosts on a network to use a particular port or ports, but first deny access for one particular host or sub-network.

B. Strategies for Rationalising the Rule Set

Firewall rule sets may be manipulated for a number of different aims. For example, it is possible to reduce the total number of rules in a rule set with the aim of reducing the potential packet-rule matching search time. Alternatively, it may be required to analyse the rules to look for, and correct, potential administrative errors.

Abbes *et al.* [3], Capretta *et al.* [4], and Liu and Gouda [5] all refer to the relationships defined by Al-Shaer and Hamed [2], when presenting their own unique methods to detect conflicts and redundancies in firewalls. These researchers make the point that too many rules can degrade the performance of a firewall by increasing the time it spends searching through unnecessary rules for a packet-rule match. According to Capretta *et al.* [4] some firewall administrators will intentionally insert some overlapping rules (technically an anomaly) so that the first rule can be made the most important. The consequence of further analysis is that it may not simply be the number of rules in a firewall policy that could cause performance degradation, but more the order they are in. An alternative strategy is presented by Liu and Gouda [6] with the "Firewall Compressor" algorithm - this can minimise the overall size of a firewall policy by reducing the number of rules. This minimisation is achieved by analysing the rules in terms of the search space they cover and then writing some new rules to cover the same space - this often results in many of the original rules being combined to produce fewer rules. The motivation of [6] came from firewall devices that have a maximum rule list size.

C. Dependency Ratio and Depth

The size of a firewall rule set, or the number of rules in the rule set, is one quantifier in measuring a rule set, however this does not determine the complexity of a rule set because it does not take into account dependency relations between rules. Hamed and Al-Shaer [7] describe two further measurements for firewall rule sets: dependency depth and dependency ratio - we have defined them in more detail below. These measurements can be used to quantify the complexity of firewall rule sets and will aid with the evaluation of our optimisation algorithm.

Here the set of all rules in a firewall is F.

Definition 1. *A* rule dependency set D *is a set of rules that have dependency relationships between the elements.* $D = \{R \in F \mid R$ *must precede or succeed another* $R_j \in F\}$ *so that for any* D_i *and* D_j *where* $i \neq j$ *then* $D_i \cap D_j = \varnothing$. *In Table I there are two such sets,* $\{R_2, R_3, R_4\}$ *and* $\{R_7, R_8\}$.

Let $Q = \{D \subset F \mid D$ a rule dependency set$\}$ be the set of all dependency sets.

Definition 2. Dependency ratio t *is the ratio of rules that must precede other rules, compared to the total number of rules.* $t = \sum_{D_i \in Q}(|D_i| - 1)/|F|$. *In Table I there are 3 rules,* $\{R_2, R_3, R_7\}$, *out of the 10 rules that must precede some other rule(s), so the dependency ratio is (2+1)/10.*

Definition 3. Dependency depth d *is the mean number of rules in dependency sets,* $d = \sum_{D_i \in Q} |D_i|/|Q|$. *In Table I there are two dependency sets,* $\{R_2, R_3, R_4\}$ *and* $\{R_7, R_8\}$, *and the dependency depth is (3+2)/2.*

III. Network Traffic Analysis

It is noted by many studies that there are distinct trends in the types of network traffic. An important parameter for this work is the distribution of port numbers and addresses. Garcia-Dorado *et al.* show that port numbers and addresses follow a Zipf distribution, although the specific parameters of the distribution vary depending upon the specific network [8]. It is this distribution, and the fact that firewall rules vary in the size of the scope of packets that they match, that allow the rule set order to be optimised. In this paper, we have used Internet trace files for the various studies and tests; these trace files have been obtained from CAIDA [9]. The actual files used are anonymised trace files taken from an OC192 Internet backbone between San Jose and Los Angeles in California, USA.

Analysis of the trace files shows that there are obvious trends in the packets. Figure 1 and Figure 2 show some results from one particular trace file. With trends like this in the Internet traffic it can be concluded that the packets will only match a small subset of a firewall rule set. This claim is backed up with studies also by Cohen and Lund [10].

Figure 1 shows the distribution of unique destination IP addresses from a chosen trace file; the graph shows just the top 200. The particular trace file used for this analysis contained 32,198,383 packets which came from 575,246 unique IP addresses. The trend in the graph shows that the majority of the IP addresses had a very small frequency. Figure 2 shows the frequency of the top 24 source port numbers from the trace file. As expected, port 80 has the highest frequency (about 14 times that of the next most frequent port). Other popular protocols can be seen here, for example Adobe RTMP, HTTP Secure, DNS, and SMTP.

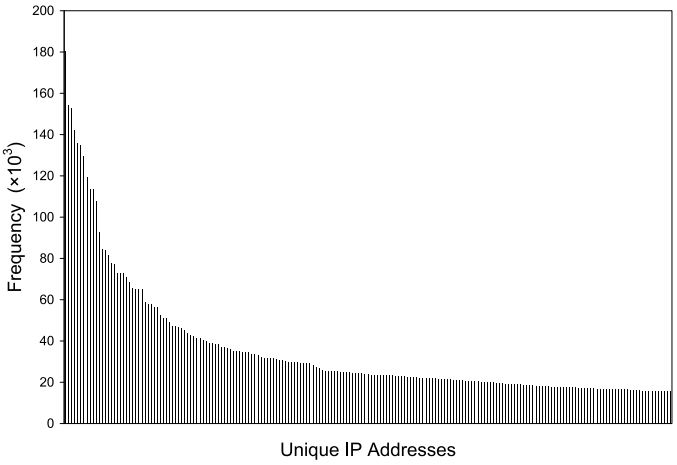

Fig. 1. The top 200 unique IP addresses rank ordered according to their frequency in the trace file. Each line represents a unique IP address.

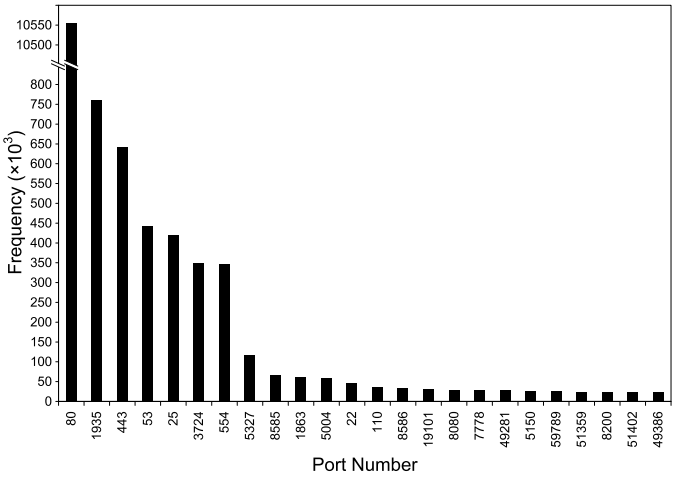

Fig. 2. The frequency of the top 24 destination port numbers ranked by frequency.

The results presented here demonstrate the motivation for changing the rule order to best suit trends in network traffic. If rules were equally distributed among the address and port space then we could conclude that these results support the argument that certain rules will match more packets than others. In practice rules are not distributed equally among the address and port space. However, evidence from firewall operators shows that there will be rules matching major services (ports) and rules matching certain exceptions for small groups of users, for example, access to certain machines for administrators only. This may in fact amplify the inequality, however, more work in this area is required.

IV. Optimisation Problem and Heuristics Solution

A. Problem Statement

The firewall optimisation problem is to place the rules in such an order so that the most frequently used rules are near to the top of the rule set and therefore reduce the packet-rule searching time. To facilitate this, the rules are associated with a weight that is equal to the number of matches of this rules in a representative flow of traffic. A naive approach would be to rank order the rules according to the weightings; but this would not take into account rule dependencies and would probably change the security policy. Therefore a more complex method is required.

The optimisation problem is very similar to the popular job scheduling problem as described by Lawler [11] and has been shown to be *NP*-Hard by [12], and [13]. Here we present a heuristic approximation algorithm that gives a good, but not necessarily optimum, solution.

For a firewall policy consisting of N filtering rules with p_i as the order (position) of rule R_i in the policy and w_i is a given weight for R_i, then the optimisation problem is to minimise a cost C which is defined as

$$C = \sum_{i=0}^{N-1} w_i p_i \tag{1}$$

s.t.

$$p_j < p_k \quad \textbf{iff } R_k \text{ is dependent upon } R_j \text{ preceding it} \tag{2}$$

Equation (1) implies that we want the rules ordered so that the largest weighted rules (i.e. those that are used the most) are near the top so that the firewall device spends less time searching down the list of rules; (2) requires that the dependency relationship be observed to comply with the required security policy. Equation (1) will be used later to measure the performance of the proposed algorithm both as part of the final evaluation and as an internal cost function.

B. Algorithm Description

The proposed approach is shown in Algorithm 1. It operates on the initial (unoptimised) rule set, `start_rules`, which contains rules associated with a weight equal to the proportion of packet matches. The starting list is immediately used to create a heap H (line 1) sorted in order of rule weight but disregarding rule dependencies. This heap has the standard functions, `HeapGet` which will find (but not remove) the highest weighted item, and `HeapRemove` which will delete a given item from any position in the heap. A second list, called `rule_list`, is also created and is initialised as empty (line 2). The `rule_list` is filled with rules as the algorithm executes and by the end will contain the final optimised rule set. This list has functions to get and remove items, as well as the ability to return pointers to the head and tail of the list. The function to insert items, `ListInsertAfter`, inserts a given item after a given position.

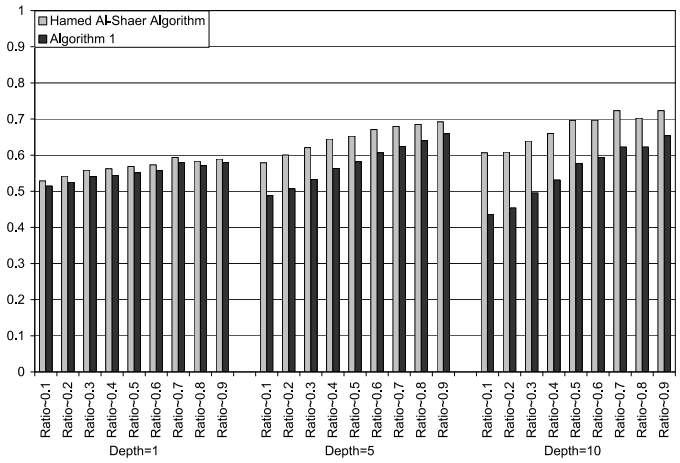

Fig. 3. Comparison between the Hamed Al-Shaer algorithm and Algorithm 1 showing the results for three different dependency depths and a range of dependency ratios. The vertical scale represents relative cost performance calculated according to (1) compared to the raw (unoptimised) rule set, a value of 1 indicates no improvement.

The outer loop (lines 3-30) continues for as long as there are items in H to be processed. The first step inside the loop is to get the highest weight rule from the heap (line 4); which for this iteration becomes what we call a *base* rule, R_b. All the rules which must precede the base rule are processed (lines 5-29) before the base rule is considered. There are two factors used to determine the position of a rule, R_c: the cost and the fact that it must come below any rules that must precede it according to (2). The `Cost` function returns the cost of a given list of rules as calculated by (1). The preceding rules are processed one by one and the cost of `rule_list` is tested with each rule in every position starting from the bottom and working up until a dependency condition means that it can go no further. The best cost determines the insertion point (line 26). It is quite possible that a preceding rule could have already been inserted in `rule_list` because it had already preceded another base rule, in this case it is ignored. When a rule is inserted into `rule_list` it is removed from the starting heap H. Once all rules preceding the base rule have been inserted, the algorithm calculates the best position for the base rule.

C. Evaluation

The algorithmic performance was evaluated in terms of cost as defined by (1) but relative to the unoptimised rule set so that relative improvement can be demonstrated. Synthetic rule sets were generated randomly using a distribution matching typical port distribution for the weight and with a range of rule dependency ratio and depth. The rule set generator created could be biased to create rule sets with a predetermined rule dependency ratio and rule dependency depth. For each dependency ratio and depth that was tested, 20 different rule sets, each with 500 rules, were generated to determine a mean performance. The performance of Algorithm 1 was compared to an earlier algorithm proposed by Hamed and Al-Shaer [7].

Algorithm 1 `OptimiseAllRules(start_rules)`

```
1:  H ← BuildHeap(H, start_rules)
2:  rule_list ← CreateList(rule_list, nil)
3:  while H ≠ ∅ do
4:    R_b ← HeapGet(H)
5:    S ← {all rules preceding R_b}
6:    for all R_c ∈ {S then R_b} do
7:      if Rc ∉ rule_list then
8:        current ← ListTail(rule_list)
9:        best_cost ← Cost(rule_list)
10:       best_pos ← current
11:       ListRemove(rule_list, R_c)
12:       while current ≠ ListHead(rule_list)
          do
13:         current ← ListPrevious(current)
14:         R_t ← ListGet(current)
15:         if R_t is not preceding R_c then
16:           ListInsertAfter(current, R_c)
17:           if  Cost(rule_list) < best_cost
              then
18:             best_cost ← Cost(rule_list)
19:             best_pos ← current
20:           end if
21:           ListRemove(rule_list, R_c)
22:         else
23:           break
24:         end if
25:       end while
26:       ListInsertAfter(best_pos, R_c)
27:       HeapRemove(H, R_c)
28:     end if
29:   end for
30: end while
31: return rule_list
```

Figure 3 shows a representative averaged set from the results with a range of rule dependency ratios (0.1 - 0.9) and depths (1, 5, and 10). Typical standard deviation on each datum point is 0.03, error bars have been omitted for clarity. The tests found that Algorithm 1 always has an improved cost performance compared to the Hamed and Al-Shaer algorithm. Analysis of the two algorithms shows that Algorithm 1 will always have an improved, or equal, performance. This is due to the fact that Algorithm 1 is guaranteed to perform a first-order check for optimum dependent rule position (within a dependency set) as is performed by the Hamed and Al-Shaer algorithm. However, Algorithm 1 performs an additional second-order check on the base rule position which at worst will not change the performance but may improve it. The best improvement for Algorithm 1 is a factor of 0.28 lower cost; this is for the highest depth, 10, and lowest ratio, 0.1. These values of dependency depth and ratio give the most freedom for the algorithm to move rules to a higher position. This corresponds to the scenario of having a small number of dependent rules but where there are a large number of rules grouped together. The smallest performance improvement, a factor of 0.017 cost reduction, was for the lowest depth, 1, and largest ratio, 0.9. This corresponds to many dependencies and with few grouped dependencies so there is less scope for Algorithm 1 to take advantage of moving rules higher in the final list.

Algorithm 1 has a worst case rule list creation performance of $N^3/4$. Hamed and Al-Shaer [7] do not give a performance indication but inspection reveals that the Hamed Al-Shaer algorithm has worst case rule list performance of $N^2/2$. The rule list creation performance of Algorithm 1 is not as good as that of the Hamed Al-Shaer algorithm because of the additional second-order check on the base rule position. In practice the run time performance, for both algorithms, is dependent on whether preceding rules stop the search for a better location for insertion of the "current" rule so that the performance is typically much better than the theoretical bound. Experimental comparison has shown that for a rule set of 500 rules (a typical rule set size) Algorithm 1 is about 11 times slower than the Hamed and Al-Shaer algorithm. With the optimisation of a static rule set this performance disadvantage is of little consequence as the optimisation of a typical rule set using Algorithm 1 is performed in the order of 10 seconds[1] and this task would only need to be performed infrequently when new rules are inserted or when a significant change in traffic performance is noted. However, if a dynamic firewall is created, and the traffic characteristics vary considerably during short time periods, then the optimisation speed may well be more important than the moderate cost improvement shown here and the algorithm proposed by [7] would be advantageous at the cost of reduced minimisation performance.

V. Conclusion

There is a motivation to optimise the rule order of firewalls such that the performance of the firewall is improved by reducing the potential number of packet-rule matches. The opportunity for optimisation arises from the fact that traffic distributions and rule ranges are not uniform; consequently some rules match many packets while others match very few. Anecdotal evidence reports that operators of firewalls typically make use of this fact and order the rules based upon knowledge of the traffic and firewall rules. There have been a few works discussing formal approaches to this and this paper adds to this small body of work. The algorithm presented here is shown to have improved performance compared to an earlier reported algorithm in all cases, at the cost of higher runtime complexity. The increased runtime complexity is unlikely to be significant for the offline optimisation of a firewall, which is the main target of this work.

Acknowledgment

Acknowledgement to Bret Giddings, University of Essex for sharing information on practical firewall configurations. This work has been funded through an EPSRC DTA award.

[1]Measurement performed on a contemporary "dual-core" PC architecture computer.

REFERENCES

[1] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Network Magazine*, vol. 15, no. 2, pp. 24–32, March/April 2001.

[2] E. Al-Shaer and H. Hamed, "Firewall policy advisor for anomaly discovery and rule editing," in *IFIP/IEEE 8th International Symposium on Integrated Network Management*, March 2003, pp. 17–30.

[3] T. Abbes, A. Bouhoula, and M. Rusinowitch, "An inference system for detecting firewall filtering rules anomalies," in *Proceedings of the 2008 ACM Symposium on Applied Computing*, ser. SAC '08. New York, NY, USA: ACM, March 2008, pp. 2122–2128.

[4] V. Capretta, B. Stepien, A. Felty, and S. Matwin, "Formal correctness of conflict detection for firewalls," in *Proceedings of the 2007 ACM Workshop on Formal Methods in Security Engineering*, ser. FMSE '07. New York, NY, USA: ACM, November 2007, pp. 22–30.

[5] A. X. Liu and M. G. Gouda, "Complete redundancy detection in firewalls," in *Data and Applications Security XIX*, ser. Lecture Notes in Computer Science, S. Jajodia and D. Wijesekera, Eds. Springer Berlin / Heidelberg, 2005, vol. 3654, pp. 193–206.

[6] A. X. Liu, E. Torng, and C. R. Meiners, "Firewall compressor: An algorithm for minimizing firewall policies," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, April 2008, pp. 176–180.

[7] H. Hamed and E. Al-Shaer, "Dynamic rule-ordering optimization for high-speed firewall filtering," in *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '06. New York, NY, USA: ACM, March 2006, pp. 332–342.

[8] J. L. García-Dorado, J. A. Hernández, J. Aracil, J. E. L. de Vergara, F. Montserrat, E. Robles, and T. de Miguel, "On the duration and spatial characteristics of internet traffic measurement experiments," *IEEE Communications Magazine*, vol. 46, no. 11, pp. 148–155, November 2008.

[9] C. Shannon, E. Aben, k. claffy, and D. Andersen, "The CAIDA anonymized 2008 internet traces." [Online]. Available: http://www.caida.org/data/passive/passive_2008_dataset.xml

[10] E. Cohen and C. Lund, "Packet classification in large ISPs: Design and evaluation of decision tree classifiers," in *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '05. New York, NY, USA: ACM, June 2005, pp. 73–84.

[11] E. L. Lawler, "Sequencing jobs to minimize total weighted completion time subject to precedence constraints," in *Algorithmic Aspects of Combinatorics*, ser. Annals of Discrete Mathematics, B. Alspach, P. Hell, and D. J. Miller, Eds. Elsevier, 1978, vol. 2, pp. 75–90.

[12] E. W. Fulp, "Optimization of network firewall policies using directed acyclic graphs," in *Proceedings of the IEEE Internet Management Conference*, 2005.

[13] W. Wang, R. Ji, W. Chen, B. Chen, and Z. Li, "Firewall rules sorting based on markov model," in *Proceedings of the The First International Symposium on Data, Privacy, and E-Commerce*, ser. ISDPE '07. Washington, DC, USA: IEEE Computer Society, November 2007, pp. 203–208.