Projection and Division: Linear-Space Verification of Firewalls

H. B. Acharya
The University of Texas at Austin
acharya @ cs.utexas.edu

M. G. Gouda
The National Science Foundation, and
The University of Texas at Austin
gouda @ cs.utexas.edu

***Abstract*—A firewall is a packet filter that is placed at the entrance of a private network. It checks the header fields of each incoming packet into the private network and decides, based on the specified rules in the firewall, whether to accept the packet and allow it to proceed, or to discard the packet. A property of a firewall is a set of packets that the firewall is required to accept or discard. Associated with each firewall is a very large set of properties that the firewall needs to satisfy. The space and time complexity of the best known deterministic algorithm, for verifying that a given firewall satisfies a given property, is $O(n^d)$, where n is the number of rules in the given firewall and d is the number of fields checked by the firewall. Usually, n is around 2000 and d is 5. In this paper, we propose the first deterministic firewall verification algorithm whose space complexity is $O(nd)$, linear in both n and d. This algorithm consists of three components: a projection pass, a division pass, and a probe algorithm. We applied our verification algorithm to over two million firewall-property pairs, varying n from 100 to 10000 and fixing d at 5. From this experiment, we observed that the algorithm requires $(900 + 0.5n)$ Kilobytes of storage and in the order of 10 seconds execution time.**

I. Introduction

A firewall is a packet filter that is placed at the point where a private computer network connects to the rest of the Internet. The firewall intercepts each packet that is exchanged between the private network and the Internet, examines the fields of the packet headers, and makes a decision to either accept the packet and allow it to proceed, or discard the packet.

The decision that a firewall makes, when it receives a packet, depends on

1) The values of the fields in the packet headers, and
2) The sequence of rules that is specified for the firewall by the firewall designer.

Each firewall rule consists of a predicate and a decision, which is either accept or discard. When the firewall receives a packet, it searches its sequence of rules for the first rule, whose predicate is satisfied by the values of the fields in the packet headers, and then applies the decision of this rule to the packet.

A discard property of a firewall F is a set of packets that F is required to discard. (The majority of the discussion in this paper focuses on discard properties of firewalls. Then in Section VI below, we show that the discussion can be easily extended to include accept properties as well.)

A firewall F is said to satisfy a discard property P iff F does discard every packet in P.

A firewall verification algorithm is an algorithm that takes as input a firewall F and a discard property P and produces as output a determination of whether F satisfies P.

Firewall verification algorithms can be classified into two classes: probabilistic and deterministic.

The time and space complexity of the best known probabilistic firewall verification algorithm [1] are $O(nd)$, where n is the number of rules in the input firewall F and d is the number of checked fields in F. Note that the value of n is about 2000, and the value of d is about 5. The problem with this probabilistic algorithm is that sometimes this algorithm produces an incorrect output (even though the analysis of [1] suggests that the probability of this algorithm producing an incorrect output is very small).

The time and space complexity of the best known deterministic firewall verification algorithm [8] are $O(n^d)$. Thus the computational complexity of this algorithm is severe given that the value of n is about 2000 and the value of d is about 5.

In this paper, we present the first deterministic firewall verification algorithm whose space complexity is $O(nd)$, linear in both n and d. Unfortunately, the time complexity of this algorithm is $O(n^{d+1})$. However, our experimental results, presented in Section VII, suggests that execution time of the algorithm takes about 10 seconds.

Our firewall verification algorithm consists of three components:

1) The projection pass
2) The division pass
3) The probe algorithm

When the projection pass is applied to any firewall F and any discard property P, it computes a new firewall, called the projection of F over P, and denoted F/P. The projection pass also returns any one of three conclusions:

(a) F satisfies P.
(b) F does not satisfy P.
(c) No conclusion can be reached.

On one hand, if the projection pass returns conclusion (a) or (b), then verifying whether F satisfies P is complete. On the other hand, if the projection pass returns conclusion (c), then we use the computed projection firewall F/P to determine whether F satisfies P.

This determination is based on the following interesting

equivalence relation, which is discussed in Section III, between F, P, and F/P:

Fact 1. *F satisfies a discard property P*
iff
F/P accepts no packets.

Therefore, to verify that F satisfies P, it is sufficient (and necessary) to verify that F/P accepts no packet. The latter can be done by applying the division pass to F/P.

When the division pass is applied to the projection firewall F/P, it divides F/P into a set of smaller firewalls called the accept slices of F/P. The following interesting equivalence relation between F/P and its accept slices is discussed in Section IV.

Fact 2. *F/P accepts no packet*
iff
every accept slice of F/P accepts no packet.

From Facts 1 and 2, we conclude that in order to verify that F satisfies a discard property P, it is sufficient (and necessary) to verify that every accept slice of F/P accepts no packet. This can be done by applying the probe algorithm, presented in Section V, to every accept slice of the projection firewall F/P.

When the probe algorithm is applied to an accept slice AS of F/P, it uses AS to generate a relatively small number of packets called probe packets of AS and proceeds to check whether AS accepts anyone of these generated probes. If AS accepts no probe packet, then we conclude that AS accepts no packet, thanks to the following equivalence relation, which is shown to hold in Section V, between packets and probe packets.

Fact 3. *An accept slice AS of F/P accepts no packet*
iff
AS accepts no probe packet.

The space complexity of each of these three components (namely the projection pass, the division pass, and the probe algorithm) is $O(nd)$, where n is the number of rules in firewall F and d is the number of fields checked in F. And because these three components constitute our firewall verification algorithm, the space complexity of our firewall verification algorithm is $O(nd)$, linear in both n and d.

II. Packets, Firewalls and Properties

In this section, we define the main terms in this paper: fields, rules, packets, firewalls, and discard properties.

A *field* is a variable whose value is taken from a nonempty interval of non-negative integers called the *domain* of the field. In this paper, we assume that there are d fields, named $f_1, .., f_d$, in the headers of each packet. (Examples of these fields are the source IP address, the destination IP address, the transport protocol, the source port number, and the destination port number.) The domain of each field f_j is denoted $D(f_j)$.

A *rule* r is of the form

$$r : f_1 \in X_1 \wedge .. \wedge f_d \in X_d \rightarrow < decision >$$

Note that r is the name of the rule, each f_j is a field, each X_j is a nonempty interval of nonnegative integers taken from the domain $D(f_j)$ of field f_j, and $< decision >$ is either accept or discard.

A rule whose decision is accept (or discard, respectively) is called an *accept rule* (or *discard rule*, respectively).

A *packet* is a tuple $(p_1, .., p_d)$ of d nonnegative integers, where each integer p_j is taken from the domain $D(f_j)$ of field f_j.

A packet $(p_1, .., p_d)$ is said to *match* a rule r of the form

$$r : f_1 \in X_1 \wedge .. \wedge f_d \in X_d \rightarrow < decision >$$

iff the predicate $(p_1 \in X_1 \wedge .. \wedge p_d \in X_d)$ is true.

A *firewall* is a sequence of rules.

A packet is said to *match* a firewall F iff the packet matches at least one rule in F.

A firewall F is called *complete* iff every packet matches F.

A firewall F is said to *accept* (or *discard*, respectively) a packet iff F has an accept (or discard, respectively) rule r such that the following two conditions hold:

1) The packet matches r
2) The packet does not match any rule that precedes r in F

Note that for any given firewall F and any given packet, exactly one of the following three statements holds:

(a) F accepts the packet
(b) F discards the packet
(c) The packet does not match F

A *discard property* P has the same form as a discard rule in a firewall:

$$P : f_1 \in Z_1 \wedge .. \wedge f_d \in Z_d \rightarrow discard$$

Note that P is the name of the property, each f_j is a field, and each Z_j is a nonempty interval of nonnegative integers taken from the domain $D(f_j)$ of field f_j.

A packet $(p_1, .., p_d)$ is said to *match* a discard property P of the form:

$$P : f_1 \in Z_1 \wedge .. \wedge f_d \in Z_d \rightarrow discard$$

iff the predicate $(p_1 \in Z_1 \wedge .. \wedge p_d \in Z_d)$ is true.

A firewall F is said to *satisfy* a discard property P iff every packet that matches P is discarded by F.

In this paper, we present an algorithm that takes as input a complete firewall F and a discard property P and determines, as output, whether or not F satisfies P. The time complexity of this algorithm is $O(n^{d+1})$ and its space complexity is $O(nd)$, where n is the number of rules in firewall F and d is the number of fields that are checked in each rule in F. Note that the space complexity of this algorithm is linear in both n and d. In practice, the value of n is about 2000 and the value of d is usually 5.

As mentioned in the Introduction, our firewall verification algorithm consists of three components:

1) The projection pass, described in Section III
2) The division pass, described in Section IV

3) The probe algorithm, described in section V

We are now ready to discuss these three components of our firewall verification algorithm in order.

III. The Projection Pass

The first step to verify whether a complete firewall F satisfies a discard property P is to apply an algorithm, called the projection pass, to both F and P. The projection pass is presented in this section. As discussed below, this algorithm performs two tasks. First, it uses F and P to compute a new firewall, usually smaller than F, called the *projection* of F over P and denoted F/P. Second, it produces any one of three conclusions:

(a) F satisfies P
(b) F does not satisfy P
(c) No conclusion can be reached.

If the produced conclusion is either (a) or (b), then the verification task is complete. But if the produced conclusion is (c), then we use the projection firewall F/P to answer the question of whether F satisfies P since, as we show below, F satisfies P iff F/P accepts no packets.

Before we can present the projection pass, we need first to introduce three new concepts:

- A firewall rule overlapping a discard property
- The projection of a firewall rule over a discard property
- A firewall rule covering a discard property

Let r be a firewall rule, and P be a discard property:

$$r : f_1 \in X_1 \wedge .. \wedge f_d \in X_d \rightarrow < \text{decision of r} >$$
$$P : f_1 \in Z_1 \wedge .. \wedge f_d \in Z_d \rightarrow discard$$

Rule r is said to *overlap* property P iff every intersection of an interval X_j in r with the corresponding interval Z_j in P is nonempty.

If rule r overlaps property P, then define the *projection* of r over P, denoted r/P, as the following rule:

$$r/P : f_1 \in (X_1 \cap Z_1) \wedge .. \wedge f_d \in (X_d \cap Z_d) \rightarrow < \text{decision of r} >$$

Note that if rule r does not overlap property P, then at least one of the intersections $(X_j \cap Z_j)$ is empty and so r/P is not a firewall rule.

Rule r is said to *cover* property P iff each interval X_j in r is a superset of the corresponding interval Z_j in P.

The projection pass is presented in Algorithm 1. Note that the time and space complexity of this pass is $O(nd)$, where n is the number of rules in firewall F, and d is the number of fields checked in every rule.

The following equivalence relation holds between a complete firewall F, a discard property P, and the projection firewall F/P that is produced by applying the projection pass to F and P.

Theorem 1. *A complete firewall F satisfies a discard property P iff the projection firewall F/P accepts no packet.*

It follows from Theorem 1 that to verify that F satisfies P, it is sufficient to verify that F/P accepts no packets.

Algorithm 1 Projection Pass

Input: A complete firewall F and a discard property P.
Output: A firewall F/P, called the projection of F over P, and any one of three conclusions:
(a) F satisfies P
(b) F does not satisfy P
(c) No conclusion can be reached

```
F/P := empty firewall
for each rule r in F do
  if r overlaps P then
    Add the rule r/P at the tail of firewall F/P
  end if
  if r covers P then
    exit loop
  end if
end for
if each rule in F/P is a discard rule then
  Terminate and declare that F satisfies P
end if
if the first rule in F/P is an accept rule then
  Terminate and declare that F does not satisfy P
else
  Declare that no conclusion can be reached
  Terminate and return F/P
end if
```

IV. The Division Pass

We verify that a projection firewall F/P accepts no packets in two steps. First, we apply an algorithm, called the division pass, to F/P to divide F/P into a set of smaller firewalls, called the accept slices of F/P. Second, we apply an algorithm, called the probe algorithm, to every accept slice of F/P to show that every slice accepts no packets. The correctness of this method is based on Theorem 2 (below) which states that F/P accepts no packet iff every accept slice of F/P accepts no packet. The division pass is presented in this section and the probe algorithm is presented in the next section.

But before we can present the division pass, we need first to introduce three new concepts:

- A firewall rule overlapping another firewall rule
- The projection of a firewall rule over another firewall rule
- A firewall rule covering another firewall rule

Let r and s be two firewall rules:

$$r : f_1 \in X_1 \wedge .. \wedge f_d \in X_d \rightarrow < \text{decision of r} >$$
$$s : f_1 \in Y_1 \wedge .. \wedge f_d \in Y_d \rightarrow < \text{decision of s} >$$

Rule r is said to *overlap* rule s iff every intersection of an interval X_j in r with the corresponding interval Y_j in s is nonempty.

If rule r overlaps rule s, then define the *projection* of r over s, denoted r/s, and the *projection* of s over r, denoted s/r,

as the following two rules:

$r/s : f_1 \in (X_1 \cap Y_1) \wedge .. \wedge f_d \in (X_d \cap Y_d) \rightarrow <$ decision of r $>$
$s/r : f_1 \in (X_1 \cap Y_1) \wedge .. \wedge f_d \in (X_d \cap Y_d) \rightarrow <$ decision of s $>$

Rule r is said to *cover* rule s iff each interval X_j in r is a superset of the corresponding interval Y_j in s.

The division pass is presented in Algorithm 2. Note that the time and space complexity of this pass is $O(nd)$, where n is the number of rules in firewall G and d is the number of fields checked in every rule.

Algorithm 2 Division Pass

Input: A firewall G.
Output: A set of firewalls $\{ AS_1, \ldots, AS_k \}$ where each AS_i is called an accept slice of G and k is the number of accept rules in G.
{Each AS_i is computed from G as follows:
AS_i := the sequence of all discard rules the precede the i-th accept rule in G, followed by the i-th accept rule in G}
for each discard rule d_r in AS_i **do**
  if d_r overlaps the accept rule a_r in AS_i **then**
    Replace d_r by d_r/a_r in AS_i
  else
    Remove d_r from AS_i
  end if
  if d_r covers the accept rule a_r in AS_i **then**
    Remove the accept rule a_r from AS_i
    Exit loop
  end if
end for

Note that each accept slice, that is computed by the division pass, consists of zero or more discard rules followed by zero or one accept rule.

The following equivalence relation holds between any firewall G and the accept slices, computed by the division pass, of G.

Theorem 2. *A firewall G accepts no packet iff every accept slice of G accepts no packet.*

From Theorems 1 and 2, to verify that a firewall F satisfies a discard property P, it is sufficient to verify that every accept slice of the projection firewall F/P accepts no packet.

V. The Probe Algorithm

In this section, we present an algorithm, called the probe algorithm, that can be applied to an accept slice AS (of some projection firewall) and determine whether AS accepts no packet. The idea of this algorithm is to identify a small set of packets, called probe packets, and check whether AS accepts no probe packet. We show in Theorem 3 below that AS accepts no packet iff it accepts no probe packet. Next, we introduce the two concepts of probe packets and probe value.

A packet $(p_1, \ldots, p_d)$ is called a *probe packet* of an accept slice AS (of some projection firewall) iff each integer p_j in the packet either equals $(b+1)$, where some discard rule in AS has a conjunct of the form "$f_j \in [a, b]$", or equals (a), where the accept rule in AS has a conjunct of the form "$f_j \in [a, b]$".

Each integer p_j in a probe packet $(p_1, \ldots, p_d)$ is called a *probe value*.

The probe algorithm is presented in Algorithm 3.

Algorithm 3 Probe

Input: An accept slice AS of a projection firewall F/P.
Output: A determination of whether AS accepts no packets.
if AS has some discard rules but no accept rule **then**
  Terminate and declare that AS accepts no packets.
end if
if AS has no discard rules and one accept rule **then**
  Terminate and declare that AS accepts at least one packet.
end if
{In all other cases, AS has some discard rules and one accept rule.}
for each field f_j **do**
  {Compute set S_j of probe values of f_j as follows:}
  $S_j := \{\}$
  for for each discard rule r in AS **do**
    if the predicate of r has the conjunct $f_j \in [a, b]$ **then**
      $S_j := S_j \cup \{b+1\}$
    end if
  end for
  if the predicate of the accept rule in AS has the conjunct $f_j \in [a, b]$ **then**
    $S_j := S_j \cup \{a\}$
  end if
end for
Compute set S of all probe packets as the Cartesian product $S_1 \times S_2 \times \ldots \times S_d$
if no probe packet in S is accepted by AS **then**
  Terminate and declare that AS accepts no packets
else
  Terminate and declare that AS accepts at least one packet
end if

Correctness of the probe algorithm is based on the following theorem.

Theorem 3. *Let AS be an accept slice (of some projection firewall) that consists of some discard rules followed by one accept rule. AS accepts no packet iff AS accepts no probe packet.*

In practice, when we implement the probe algorithm we do not generate and store set S of all probe packets, because this set has $O(n^d)$ packets in the worst case. Instead, we use nested loops to traverse the d sets $S_1, S_2 ... S_d$ and generate the probe packets, one by one, and check whether each generated packet is accepted by AS before generating the next packet. This modification does not affect the correctness of the algorithm, but reduces the space required from $O(n^d)$ to $O(nd)$. Unfortunately, the time complexity of the probe algorithm is $O(n^d)$.

VI. Verifying Accept Properties

In Sections III, IV, and V, we presented the three components of of an algorithm for verifying whether a firewall F satisfies a discard property P. This presentation can be extended in a straightforward manner to verifying whether a firewall F satisfies an accept property P. The extension consists of replacing each occurrence of "discard" with an occurrence of "accept" and vice versa.

For example, Theorems 1, 2, and 3 (above), that can be used to verify discard properties, can be extended to Theorems 4, 5, and 6 (below), that can be used to verify accept properties.

Theorem 4. *A firewall F satisfies an accept property P iff the projection firewall F/P discards no packet.*

Theorem 5. *A firewall G discards no packet iff every discard slice of G discards no packet.*

Theorem 6. *Let DS be a discard slice (of some projection firewall) that consists of some accept rules followed by one discard rule. DS discards no packet iff DS discards no probe packet.*

VII. Experimental Results

It is easy to see that the asymptotic space complexity of our algorithm is linear in n (and d). However, the probe algorithm requires that a slice be tested against a large number of packets. A slice may require $O(n^d)$ probe packets in the worst case, and there may be up to $O(n)$ slices. Thus, the time complexity of the algorithm is $O(n^{d+1})$, which is pseudo-polynomial. There are several natural questions about the algorithm:

- Does the linear space requirement hold in practice?
- Is the algorithm fast enough that it can be used for practical verification of firewalls?
- Does the algorithm scale well if we employ it to verify large firewalls?

In this section, we demonstrate that the answer to all these questions is "yes".

A. Experiment 1

In this experiment, we generate firewalls with n rules each, where n is varied from 100 to 10000 in steps of 100. d has the standard value 5. For each value of n, we generate 100 random firewalls, and for each firewall, we verify 100 randomly generated properties. We thus verify one million firewall-property pairs using our algorithm.

The maximum observed space requirements for this algorithm are shown in Figure 1.

The space consumption is not only linear, as expected, it also grows very slowly at a steady rate of $40 - 50$ kilobytes per 100 rules. We suggest that under the conditions of our experiment, an upper bound for the memory consumption is $900 + 0.5n$ kilobytes.

# Rules	Time (sec.)
9800	18.3
10000	18.2
9900	18.1
9500	17.6
9400	17.0

TABLE I
Five longest running times (seconds)

B. Experiment 2

In this experiment, we again verify one million firewall-property pairs (as in Experiment 1), and note the running time required by our algorithm.

The execution time of our complete algorithm, consisting of the projection pass, the division pass, and the probe algorithm, are presented in Figure 2. We give both the average and the maximum execution time observed for firewalls of a given length.

The speed of the algorithm enables us to verify a very large number of firewall-property pairs, so the graph shows a clear trend. The worst-case running time is a very modest 18.3 seconds. Indeed, for firewalls of length up to 2000, which is a good bound for practical firewalls, the worst-case time is 0.71 seconds.

Table I shows the speed of our algorithm. As expected, the worst execution times were for large firewalls with close to $10,000$ rules. However, even in these pathological cases, our algorithm had an execution time of the order of 10 seconds. It may be noted that all experiments were run on a 3.06 GHz processor, using only one core; as the algorithm requires probing of many slices, which are independent tasks, it is simple to further speed up its execution using a parallel computer. Furthermore, our implementation of the algorithm is an interpreted script; an efficient compiled version in C/C++ should also improve performance considerably.

From the two million firewall-property pairs we verify in Experiments 1 and 2, we can say with high confidence that despite the pseudo-polynomial running time, in practice our algorithm takes very little time to complete, and is suitable for practical firewall verification.

C. Experiment 3

In this experiment, we attempted to measure whether our algorithm scales well with the size of the problem instance. Our experiment involved "torture testing" the algorithm with extremely large inputs, two or three orders of magnitude larger than practical firewalls. Our results remained positive.

For 100 firewall-property pairs (10 firewalls and 10 random properties each) where the firewall length was 100000, the space required was 44 MB, as opposed to the 50MB predicted by our bound. The mean time was 66.5 seconds and the maximum time 842 seconds - about 14 minutes.

Fig. 1. Results of Experiment 1.

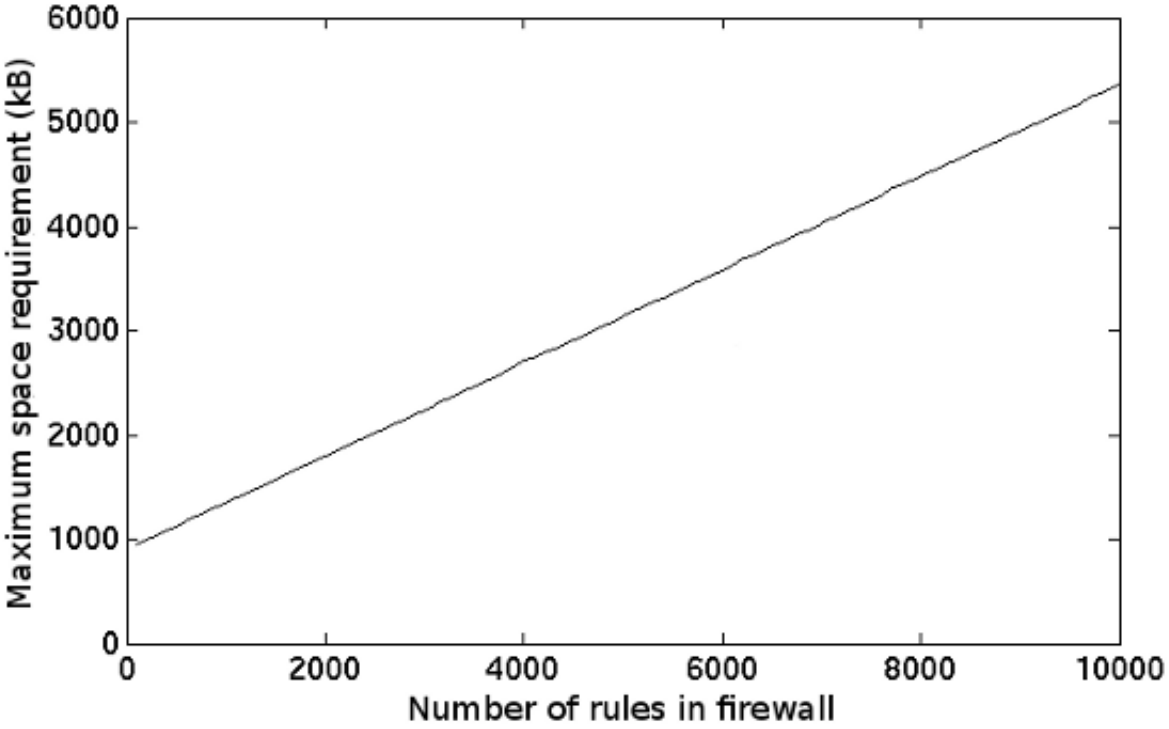
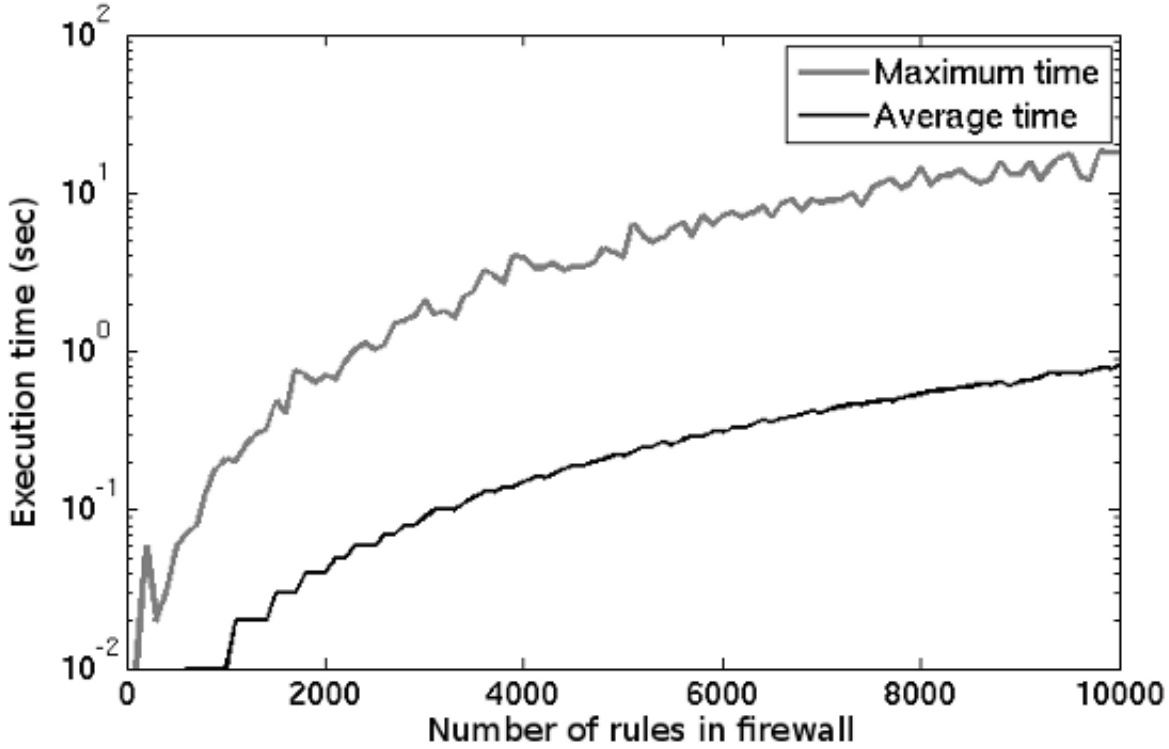


Fig. 2. Results of Experiment 2.



For 100 (again 10×10) firewall-property pairs with one million rules, the mean time was 2647 seconds (approx. 44 minutes), and the maximum time 66261 seconds ($\approx$ 18.5 hours), but the maximum memory consumption was a very reasonable 436MB - considerably less than the 500MB suggested by the bound.

VIII. Related Work

Firewalls are a critical component of network security, so it is natural that there has been considerable research attacking the problem of how to ensure a firewall is truly secure. In this section, we briefly discuss the previous approaches to the problem of ensuring firewall correctness.

1) *Firewall Testing:* To test a given firewall F, one generates packets for which the "expected" decisions of F, accept or discard, are known a priori. The generated packets are then sent to F, and the actual decisions of F for these packets are observed. If the expected decision for each generated packet is the same as the actual decision for the packet, one concludes that the given firewall F is probably correct. Otherwise, the given firewall F has errors. Al-Shaer et al. provide a complete framework to generate targeted packets and obtain good coverage in testing in [3].
2) *Firewall Analysis:* A firewall may have various kinds of defects: vulnerabilities, conflicts, anomalies, and redundancies. Firewall analysis refers to the use of algorithms to identify these defects in a given firewall F.
   The concept of conflicts between rules in a firewall is due to [5] and [2]. A classification of anomalies, as well as algorithms to detect them, is presented in [4].A framework for understanding the vulnerabilities in a single firewall is outlined in [6], and an analysis of these vulnerabilities presented in [9]. [13] is a quantitative study of configuration errors for a firewall. An example of an efficient firewall analysis algorithm is given in FIREMAN [14]. An integrated analysis engine for firewalls in a network is given in Fang [12].
3) *Firewall Verification:* A verification algorithm takes a firewall F and a property R, and determines whether or not the firewall F satisfies the property R. The question of how to query a given firewall and obtain the answer (whether or not it satisfies a given property) is discussed in [12] and [11].
4) *Firewall Design:* To ensure a firewall does not have vulnerabilities or other problems, it can be designed from the outset using structured algorithms. Such algorithms, that can generate a firewall from its specification, are provided in [7] and [10].

The algorithm presented in this paper is a firewall verification algorithm, and belongs in group 3 of the types of algorithm presented above. It is, in fact, the most efficient firewall verification algorithm known.

IX. Conclusion

This paper addresses the important practical problem of verifying whether a given firewall satisfies a property. Current deterministic algorithms that solve this problem have time and space complexity of $O(n^d)$. In contrast, our approach takes only $O(nd)$ space, which, being the size of the given firewall, is clearly a lower bound on the space complexity. However, our worst-case running time is $O(n^{d+1})$. By verifying millions of firewall-property pairs, we show that in spite of the worst-case pseudo-polynomial running time, the algorithm is fast enough to use for practical firewall verification.

In this paper, we make three new contributions. Our first contribution is a technique which we call firewall projection, which prunes a given firewall F down to a smaller firewall F/P using the given property P. F/P consists of the portion of F that interacts with packets that match property P, so F/P satisfies P iff F satisfies P.

Our next contribution, firewall division, divides the firewall F/P into multiple smaller firewalls which we call "slices". F/P satisfies P iff every slice satisfies P.

The final contribution of this paper is the probe algorithm, a new verification algorithm that takes as input a slice F and a property P, and decides whether F satisfies P. This algorithm takes only $O(nd)$ space; the worst-case time is $O(n^{d+1})$. While the theoretical worst-case running time seems to be large, we have extensively demonstrated that the algorithm is in fact very usable for practical firewalls. (In the worst case, it still terminates in less than a second for firewalls of 2000 rules. The average time for such a firewall is 40 milliseconds.)

At the moment, this work deals with single firewalls only, but it can be generalized to any case where a policy is expressed as a list of rules with first-match semantics. We can also use this algorithm in conjunction with the techniques described in [8] in order to verify enterprise networks with tens or even hundreds of firewalls. We suggest that exploring whether our algorithm is useful in other domains - such as verifying access control lists in general - may be an interesting area for further research.

It is also noteworthy that verifying whether a given firewall satisfies a given property can be used as a basic step in performing other tasks - for instance, checking whether a particular rule in a given firewall is redundant. We believe that our algorithm may be useful for developing an efficient redundancy-detection algorithm, and will attempt to develop a solution to this problem in our future work.

References

[1] H. B. Acharya and M. G. Gouda. Linear-time verification of firewalls. In *Proceedings of the International Conference on Network Protocols*, 2009.

[2] H. Adiseshu, S. Suri, and G. M. Parulkar. Detecting and resolving packet filter conflicts. In *INFOCOM*, pages 1203–1212, 2000.

[3] E. S. Al-Shaer, A. El-Atawy, and T. Samak. Automated pseudo-live testing of firewall configuration enforcement. *IEEE Journal on Selected Areas in Communications*, 27(3):302–314, 2009.

[4] E. S. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan. Conflict classification and analysis of distributed firewall policies. *IEEE Journal on Selected Areas in Communications*, 23(10):2069–2084, 2005.

[5] D. Eppstein and S. Muthukrishnan. Internet packet filter management and rectangle geometry. In *SODA*, pages 827–835, 2001.

[6] M. Frantzen, F. Kerschbaum, E. E. Schultz, and S. Fahmy. A framework for understanding vulnerabilities in firewalls using a dataflow model of firewall internals. *Computers & Security*, 20(3):263–270, 2001.

[7] M. G. Gouda and A. X. Liu. Strucured firewall design. *Computer Networks*, 51:1106–1120, 2007.

[8] M. G. Gouda, A. X. Liu, and M. Jafry. Verification of distributed firewalls. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, 2008.

[9] S. Kamara, S. Fahmy, E. E. Schultz, F. Kerschbaum, and M. Frantzen. Analysis of vulnerabilities in internet firewalls. *Computers & Security*, 22(3):214–232, 2003.

[10] A. X. Liu and M. G. Gouda. Diverse firewall design. *IEEE Transaction on Parallel and Distributed Systems*, 19(9):1237–1251, 2008.

[11] A. X. Liu and M. G. Gouda. Firewall policy queries. *IEEE Transactions on Parallel and Distributed Systems*, 20(6):766–777, 2009.

[12] A. J. Mayer, A. Wool, and E. Ziskind. Fang: A firewall analysis engine. In *IEEE Symposium on Security and Privacy*, pages 177–187, 2000.

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

[14] L. Yuan, J. Mai, Z. Su, H. Chen, C.-N. Chuah, and P. Mohapatra. Fireman: A toolkit for firewall modeling and analysis. *Security and Privacy, IEEE Symposium on*, 0:199–213, 2006.

View publication stats