Linear-Time 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 filter placed at the entrance of a private network. Its function is to examine each packet that is incoming into the private network and decide, based on the specified rules of the firewall, whether to accept the packet and allow it to proceed, or to discard the packet. A property of a firewall is a specified set of packets that is supposed to be accepted or discarded by the firewall. In this paper, we present the first linear time algorithm to verify whether a given firewall satisfies a given property. The time complexity of our algorithm is $O(nd)$, where n is the number of rules in the given firewall and d is the number of fields that are checked by the firewall. Our verification algorithm consists of two passes: a deterministic pass followed by a probabilistic pass. In most cases, the algorithm correctly determines whether the given firewall satisfies the given property. But in some rare cases, the algorithm may erroneously determine that the firewall satisfies the property. Using a combination of analysis and extensive simulation, we show that the probability of an error by the algorithm is of the order of 6×10^{-5}.**

I. Introduction

A firewall is a security system that acts as a protective boundary between a private computer network and the "outside world", usually the rest of the Internet. Firewalls help make private computer networks invisible to online attackers .

A firewall is a packet filter that is placed at a point where a private computer network connects to the Internet. The firewall intercepts each packet that is exchanged between the network and the Internet, examines the headers of the packet, and based on a sequence of rules in the firewall, makes a decision concerning the packet: it may "accept" the packet and allow it to proceed, or it may "discard" the packet.

The decision that a firewall makes when it receives a packet depends on the values in the headers of the packet. A firewall is a sequence of rules; each rule matches certain packets, with specific header values, and specifies the decision (accept or discard) to be made for that packet. In case there are multiple rules matching a packet, the first rule (according to the order of the rule sequence), that matches the packet, takes precedence.

A firewall property is specified by a set of packets and a decision (accept or discard) that needs to be made by the firewall for all the packets in the specified set. A firewall is said to satisfy a property iff the decision made by the firewall for all the packets specified by the property matches the decision specified by the property.

In this paper, we present a firewall verification algorithm. This algorithm takes a firewall and a property, and determines whether or not the firewall satisfies the property. The time complexity of our algorithm is $O(nd)$, where n is the number of rules in the given firewall, and d is the number of fields checked by each rule in the given firewall. Thus the time complexity of this algorithm is linear in both n and d. By way of contrast, the fastest prior algorithm for firewall verification [14], [9] has a time complexity of $O(n^d)$ [13].

In an industrial setting, a firewall has about $2,000$ rules, each rule checks between 5 and 7 fields, and a firewall needs to be verified against $10,000$ or more properties. Thus, it is a great advantage to use our algorithm, instead of any of the previous algorithms, to verify firewalls in industrial settings.

Our verification algorithm in this paper takes as input a firewall F and a property P, and decides whether F satisfies P. The algorithm consists of two passes: a deterministic pass and a probabilistic pass.

The deterministic pass produces any one of three outcomes:

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

The probabilistic pass, which is applied only if the deterministic pass produces outcome (c), produces any one of two outcomes:

- (c_1) F satisfies P with a high probability.
- (c_2) F does not satisfy P.

Thus, our firewall verification algorithm returns any one of four outcomes: $(a), (b), (c_1)$ and (c_2). Outcomes $(a), (b)$ and (c_2) are guaranteed to be correct. Only outcome (c_1) can possibly be erroneous. We show, by analysis, that the probability of outcome (c_1) being erroneous is around 0.6%.

We have implemented our algorithm and applied it extensively to verify whether randomly-generated firewalls satisfy given properties. Our experiments, using $80,000$ firewall-property pairs, have shown that our verification algorithm returns outcome c_1 less than 1% of the time. Therefore, we conclude that the probability of our verification algorithm producing an erroneous result is less than $.006 \times .01 = 6 \times 10^{-5}$, which is vanishingly small.

II. Firewalls, Packets and Properties

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

A *field* is a variable, whose value is taken from an interval of non-negative integers. Examples of fields are source IP address, destination IP address, transport protocol, source port number, and destination port number. The domain of values of the source IP address field, for example, is the interval $[0, 2^{32} - 1]$.

978-1-4244-4634-6/09/$25.00 ©2009 IEEE

In this paper, we consider d fields, denoted $f_1, f_2, \ldots, f_d$. The domain of values of any field f_j is denoted by $D(f_j)$. Note that $D(f_j)$ is an interval of non-negative integers.

A *firewall* F is a sequence of n rules, where the i-th rule is of the form:

$$f_1 \in S_{i,1} \wedge f_2 \in S_{i,2} \wedge \ldots \wedge f_d \in S_{i,d} \rightarrow < decision_i >$$

Each $S_{i,j}$ is a non-empty interval of non-negative integers taken from the domain $D(f_j)$, and $< decision_i >$ is either *accept* or *discard*. The last rule in a firewall is of the form

$$f_1 \in S_{n,1} \wedge f_2 \in S_{n,2} \wedge \ldots \wedge f_d \in S_{n,d} \rightarrow < decision_n >$$

Each $S_{n,j}$ is the domain $D(f_j)$ of field f_j.

A *packet* is a d-tuple $(p_1, p_2, \ldots, p_d)$, where each p_j is an element from the domain $D(f_j)$ of field f_j .

A packet $(p_1, p_2, \ldots, p_d)$ is said to *match* a rule

$$f_1 \in S_{i,1} \wedge f_2 \in S_{i,2} \wedge \ldots \wedge f_d \in S_{i,d} \rightarrow < decision_i >$$

in a firewall F iff the predicate $p_1 \in S_{i,1} \wedge p_2 \in S_{i,2} \wedge \ldots \wedge p_d \in S_{i,d}$ holds.

A firewall F is said to *accept*, or *discard* respectively, a packet iff F has a rule that satisfies the following three conditions:

1) The packet matches the rule.
2) The packet matches no earlier rule in F.
3) The decision of the rule is accept, or discard, respectively.

A property is of the form:

$$f_1 \in T_1 \wedge f_2 \in T_2 \wedge \ldots \wedge f_d \in T_d \rightarrow < decision >$$

Each T_j is a non-empty interval of non-negative integers taken from the domain $D(f_j)$ of field f_j, and $< decision >$ is either *accept* or *discard*.

A packet $(p_1, p_2, \ldots, p_d)$ is said to *match* a property

$$f_1 \in T_1 \wedge f_2 \in T_2 \wedge \ldots \wedge f_d \in T_d \rightarrow < decision >$$

iff the predicate $p_1 \in T_1 \wedge p_2 \in T_2 \wedge \ldots \wedge p_d \in T_d$ holds.

A firewall F is said to *satisfy* a property

$$f_1 \in T_1 \wedge f_2 \in T_2 \wedge \ldots \wedge f_d \in T_d \rightarrow < decision >$$

iff one of the following two conditions holds:

1) $< decision >= accept$, and every packet that matches the property is accepted by F .
2) $< decision >= discard$, and every packet that matches the property is discarded by F .

A firewall example F_1, with three rules, is given below:

$$f_1 \in [4, 10] \wedge f_2 \in [5, 7] \wedge f_3 \in [4, 10] \rightarrow \textit{accept}$$
$$f_1 \in [1, 6] \wedge f_2 \in [5, 7] \wedge f_3 \in [5, 7] \rightarrow \textit{accept}$$
$$f_1 \in [0, 10] \wedge f_2 \in [0, 10] \wedge f_3 \in [0, 10] \rightarrow \textit{discard}$$

Each rule checks the three fields f_1, f_2, and f_3 . The domain $D(f_j)$ of each field f_j is the interval $[0, 10]$. A packet for firewall F_1 is the 3-tuple $(1, 7, 5)$, where $f_1 = 1$, $f_2 = 7$, and $f_3 = 5$. This packet does not match the first rule in F_1, but matches the second rule in F_1, and because the decision of the second rule is *accept*, firewall F_1 is said to accept the packet.

Three properties of firewall F_1, named P_1 through P_3, are as follows:

$$P_1 : f_1 \in [3, 9] \wedge f_2 \in [8, 10] \wedge f_3 \in [6, 9] \rightarrow \textit{discard}$$
$$P_2 : f_1 \in [2, 3] \wedge f_2 \in [1, 9] \wedge f_3 \in [6, 9] \rightarrow \textit{discard}$$
$$P_3 : f_1 \in [1, 9] \wedge f_2 \in [4, 9] \wedge f_3 \in [3, 7] \rightarrow \textit{accept}$$

Note that packet $(1, 7, 5)$ matches P_3, but does not match P_1 or P_2 .

In this paper, we present an efficient algorithm for verifying whether a given firewall F satisfies a given property P. This algorithm consists of two passes: a deterministic pass followed, if necessary, by a probabilistic pass.

The deterministic pass, described in Section III below, is applied first to the given firewall F and property P. This pass produces any one of the following three outcomes:

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

If the deterministic pass produces outcome (c), then it will also produce a reduced version F' of firewall F, called a "projection" of F over property P. Both F' and P are then input to the probabilistic pass, described in Section V below. This pass produces either one of the following two outcomes:

(c_1) F satisfies P (with high probability).
(c_2) F does not satisfy P.

III. Deterministic Pass

The deterministic pass is based on the following three concepts:

- a rule r in a firewall F "overlaps" a property P
- a rule r in a firewall F "covers" a property P
- a "projection" of a firewall F over a property P

We define these three concepts next.

Let r be a rule in a firewall F and let P be a property of F. Rule r is said to *overlap* property P iff there exists a packet of F that matches both rule r and property P.

Let r be a rule in a firewall F and let P be a property of F. Rule r is said to *cover* property P iff every packet, that matches property P, also matches rule r.

Let F be a firewall and let P be a property of F. A *projection* F' of F *over* P is a firewall that satisfies the following two conditions:

1) F' is constructed from F by removing zero or more rules, other than the last rule, from F.
2) F' satisfies P iff F satisfies P.

Note that although F itself can be considered a projection of F over P, we are interested in a projection F' that has a smaller number of rules than F. This is because, if F' has a smaller number of rules than F, then it is easier to check "whether F' satisfies P" than to check "whether F satisfies

P" and use the conclusion of "whether F' satisfies P" to determine "whether F satisfies P".

The deterministic pass proceeds as follows. First, it starts at the beginning of the given firewall F and identifies all the rules that overlap the given property P, one by one, until it encounters a rule that covers P. Second, the deterministic pass produces one of the following three outcomes:

(a) If all the rules in F, that have been identified to overlap property P, have the same decision as that of P, then report that F satisfies P.
(b) Else if the first rule in F, identified to overlap P, has a different decision than that of P, then report that F does not satisfy P.
(c) Else report that no conclusion can be reached and return the following projection F' of firewall F over property P. Projection F' consists only of all the rules in F, that have been identified to overlap P, followed by the last rule in F if this rule has not been identified to overlap P.

The deterministic pass is shown in Algorithm 1.

Applying the deterministic pass to firewall F_1 in Section II and to each of the properties P_1, P_2 and P_3 specified in Section II, we conclude that F_1 satisfies P_1, F_1 does not satisfy P_2, and no conclusion can be reached on whether F_1 satisfies P_3 .

Theorem 1. *The time complexity of the deterministic pass, when applied to a firewall F and a property P, is $O(nd)$, where n is the number of rules in F and d is the number of fields in a rule (as well as in P).* □

IV. Effectiveness of the Deterministic Pass

In this section we identify three classes of firewalls and properties for which the deterministic pass is guaranteed to produce either outcome (a) or outcome (b), but never outcome (c). Thus, the probabilistic pass is not needed for verifying these classes of firewalls and properties.

A. Singleton Properties

A *singleton property* is one that is only matched by a single packet, i.e. this property is of the form

$$f_1 \in T_1 \ldots \wedge f_d \in T_d \rightarrow < decision >$$

where each interval T_j contains a single non-negative integer t_j. In this case, this property can also be expressed as

$$f_1 = t_1 \wedge \ldots \wedge f_d = t_d \rightarrow < decision >$$

Theorem 2. *If the deterministic pass is applied to a firewall F and a singleton property P, then the algorithm is guaranteed to produce either outcome (a) or outcome (b).* □

This result is used in the probabilistic pass (of our firewall verification algorithm) presented in Section V.

Algorithm 1 Deterministic Pass

Input: A firewall F and a property P of F .
Output: Any one of the following outcomes:
(a) F satisfies P.
(b) F does not satisfy P.
(c) No conclusion can be reached, and, in this case, produce a projection F' of F over P.

Variables:
i : an index variable in range $1..(n+1)$, initially 1.
v : an array of n boolean elements, initially $false$.

while $i \leq n$ **do**
  if r_i overlaps P **then**
    $v[i] = true$
  end if
  if r_i covers P **then**
    $i := n + 1$
  else
    $i := i + 1$
  end if
end while

if for every j in range $1..n$, where $v[j] = true$, decision of r_j = decision of P **then**
  report F satisfies P
else if for smallest k in range $1..n$, where $v[k] = true$, decision of $r_k \neq$ decision of P **then**
  report F does not satisfy P
else
  report no conclusion can be reached and return the following projection F' of firewall F over property P. Projection F' consists only of all the rules in F, that have been identified to overlap P, followed by the last rule in F if this rule has not been identified to overlap P.
end if

B. Two-phase firewalls

A firewall F is called *two-phase* iff each rule, other than the last rule, in F has a decision that is different from that of the last rule in F.

Theorem 3. *If the deterministic pass is applied to a two-phase firewall F and a property P, whose decision is the same as that of the last rule in F, then the algorithm is guaranteed to produce either outcome (a) or outcome (b).* □

This result is important because, in many practical cases, a firewall is designed by specifying all the rules that accept packets, followed by a default rule that discards all (remaining) packets. Any such firewall is a two-phase firewall.

C. Conflict-free Firewalls

A firewall F is said to be *conflict-free* iff, for any two rules r_i and r_j, other than the last rule, in F, at least one of the

following two conditions holds:

1) No packet matches both r_i and r_j.
2) r_i and r_j have the same decision.

Theorem 4. *If the deterministic pass is applied to a conflict-free firewall F and a property P, whose decision is the same as that of the last rule in F, then the algorithm is guaranteed to produce either outcome (a) or outcome (b).* □

Note that each two-phase firewall is also conflict-free. Thus, correctness of Theorem 3 follows from the correctness of Theorem 4.

V. Probabilistic Pass

In the previous section, we discussed several instances of firewall F and property P, where the deterministic pass applied to F and P produces either outcome (a) or outcome (b). The deterministic pass, however, can produce outcome (c) in many other instances of F and P. In order to handle these cases, we perform the probabilistic pass, which we detail in this section.

Theorem 5. *Any property P can be decomposed into an equivalent set Q of singleton properties, such that any firewall F satisfies P iff F satisfies each singleton property in Q.* □

We refer to set Q in Theorem 5 as the *footprint* of property P. Note that, for every packet p that matches P, there is a *corresponding* singleton property in Q that is matched by p.

As an example, assume that we need to verify whether or not a given firewall F satisfies the following property P:

$$f_1 \in [1,2] \wedge f_2 \in [5,5] \wedge f_3 \in [1,2] \rightarrow discard$$

By Theorem 5, firewall F satisfies P iff F satisfies each of the following singleton properties in the footprint of P:

$$f_1 = 1 \wedge f_2 = 5 \wedge f_3 = 1 \rightarrow discard$$
$$f_1 = 1 \wedge f_2 = 5 \wedge f_3 = 2 \rightarrow discard$$
$$f_1 = 2 \wedge f_2 = 5 \wedge f_3 = 1 \rightarrow discard$$
$$f_1 = 2 \wedge f_2 = 5 \wedge f_3 = 2 \rightarrow discard$$

By Theorem 2, the deterministic pass can be used to verify that F satisfies each of these four singleton properties, producing either outcome (a) or outcome (b).

It follows from this example that we can determine whether a firewall F satisfies a property P by decomposing P into the singleton properties of its footprint and verifying whether F satisfies each of these singleton properties (using the deterministic pass).

However, the number of singleton properties in the footprint of a property can be very large. For example, there are 40000 singleton properties in the footprint of the following property:

$$f_1 \in [1,200] \wedge f_2 \in [5,5] \wedge f_3 \in [1,200] \rightarrow discard$$

To get around this problem, we selectively verify some of the singleton properties in the footprint of P, not all of them. In our probabilistic pass discussed below, we verify K distinct singleton properties, where K is a positive integer chosen by the human verifier. (Below, we argue that the value of K can be around 1024). Next, we present several concepts that can be used in selecting the K singleton properties.

Let P be a property of some firewall that has d fields: $f_1, \ldots, f_d$. Then P can be written as

$$f_1 \in [a_1, b_1] \wedge \ldots \wedge f_d \in [a_d, b_d] \rightarrow < decision >$$

where for each field f_j, the two values a_j and b_j are called the *corner values* of f_j in P. Note that each field f_j is either 2-*cornered in* P, if $a_j \neq b_j$, or 1-*cornered in* P, if $a_j = b_j$.

A packet $p = (p_1, p_2, \ldots, p_d)$, that matches property P, is called a *corner packet of* P iff each value p_j in packet p is a corner value of field f_j in P.

We would like to choose the K singleton properties to correspond, as much as possible, to the corner packets of P. Unfortunately, the number of corner packets of P can be exponential in the number of fields d in the worst case, as shown by the following theorem.

Theorem 6. *Let P be a property of some firewall that has d fields. If the number of $2-$cornered fields in P is k, where $k \leq d$, then the number of corner packets of property P is 2^k.* □

It follows from this theorem that if $K \leq 2^k$ then we can choose any K corner packets and compute the K singleton properties that correspond to them. On the other hand, if $K > 2^k$, then we can choose all the 2^k corner packets, plus $K - 2^k$ other packets that match P, and compute the K singleton properties that correspond to them.

The probabilistic pass is shown in Algorithm 2.

Theorem 7. *The time complexity of the probabilistic pass, when applied to a firewall F, a property P, a projection F' of F over P, and a constant K, is $O(nd)$, where n is the number of rules in F and d is the number of fields checked in F (as well as in P).* □

Note that, in the probabilistic pass, if K is chosen to be at least 2^k, then set $pset$ contains every corner packet of the given property P, and the probabilistic pass is much more likely to produce accurate results. Now, to ensure that $K \geq 2^k$ in most cases, we observe that $d \geq k$, and d does not exceed 10 in most cases. Thus, selecting K to be 1024 ensures that $K \geq 2^k$ in most cases. In the next section, we show that the value $K = 1024$ is ample for the probabilistic pass to achieve high accuracy.

VI. Effectiveness of the Probabilistic Pass

In this section, we show that when the probabilistic pass reports that the given firewall F satisfies the given property P, then the probability that F actually satisfies P is high.

Specifically, let A be the event that F satisfies P, and let B be the event that the probabilistic pass reports that F satisfies P. Thus, we need to prove that the conditional probability $P(A|B)$ is high. A derivation of the conditional probability

Algorithm 2 Probabilistic Pass

Input: A property P of a firewall F, a projection F' of F over P, and a constant K.

Output: Any one of the following outcomes:

(c_1) F satisfies P with a high probability. (See Theorem 8 below.)

(c_2) F does not satisfy P.

1: Let k be the number of 2-cornered fields in P. (Note that $k \leq d$ where d is the total number of fields in P or F.)
2: **if** $K \leq 2^k$ **then**
3: select any K corner packets of P and put them in a set named $pset$.
4: **else**
5: put all the 2^k corner packets of P in set $pset$ and add any $(K - 2^k)$ packets, that match property P, to set $pset$.
6: **end if**
7: For each packet $(p_1, \ldots, p_d)$ in $pset$, compute a corresponding singleton property as follows:

$$f_1 = p_1 \wedge \ldots \wedge f_d = p_d \rightarrow < decision >$$

where $< decision >$ is the decision of the given property P.
8: Let $qset$ be the set of singleton properties thus computed. Note that set $qset$ is a subset (usually proper) of the footprint of property P.
9: Use the deterministic pass to verify whether or not the given projection F' satisfies every singleton property in set $qset$.
10: If F' satisfies every singleton property in set $qset$, then report that F satisfies property P with high probability. Otherwise, report that F does not satisfy property P.

$P(A|B)$ follows:

$$\begin{aligned} P(A|B) &= \frac{P(AB)}{P(B)} \\ &= \frac{P(AB)}{P(AB) + P(\bar{A}B)} \\ &= \frac{P(A)}{P(A) + P(\bar{A}B)} \text{ , since } P(AB) = P(A) \\ &= \frac{P(A)}{P(A) + P(\bar{A})P(B|\bar{A})} \\ &= \frac{P(A)}{P(A) + (1 - P(A))P(B|\bar{A})} \end{aligned}$$

Thus, to compute $P(A|B)$, we need to estimate the two quantities $P(A)$ and $P(B|\bar{A})$.

First, we assume that $P(A)$ has the unbiased value of $\frac{1}{2}$. (Note that there are two possibilities - either the given firewall satisfies the given property, or not. Thus, this assumption is based on the view that these two possibilities are equally likely.) In practice, the value of $P(A)$ is close to 1, but we assume that its value is $\frac{1}{2}$ and show that the algorithm is still highly reliable, even under this severe assumption.

Second, we estimate that the value of $P(B|\bar{A})$ is at most E^K, where E is the fraction of singleton properties in the footprint of P that are satisfied by F,and K is the number of singleton properties that are selected from the footprint of P and shown to be satisfied by F in the probabilistic pass.

It follows that

$$\begin{aligned} P(A|B) &\geq \frac{\frac{1}{2}}{\frac{1}{2} + (1 - \frac{1}{2})E^K} \\ &= \frac{1}{1 + E^K} \end{aligned}$$

This derivation proves the following theorem:

Theorem 8. *If the probabilistic pass reports that F satisfies P, then the probability that F satisfies P is at least $\frac{1}{1+E^K}$.* □

In practice, we use the constant value of $K = 1024$. Thus, even if the value of E is 99.5%, the probability $P(A|B)$ is at least 99.41%. (In the more practically reasonable case where $P(A) = 0.99$, $P(A|B)$ rises to over 99.99%.)

VII. Experimental Results

In the previous sections, we have presented a linear time firewall verification algorithm, whose time complexity is $O(nd)$, where n is the number of rules in the given firewall and d is the number of fields checked in the firewall (or the given property). The verification algorithm consists of a deterministic pass followed by a probabilistic pass (which is run only on the cases where the deterministic pass fails to decide whether or not the given firewall satisfies the given property). Still, three important questions remain to be answered. They are:

1) What is the speed of a working implementation of our verification algorithm?
2) The deterministic pass runs 1000 times faster than the probabilistic pass. What percentage of properties is decided by the deterministic pass alone?
3) There is a probability of error in the case where the probabilistic pass produces outcome (c_1). How frequent are these cases?

A. Experiment 1

In our first experiment, we generated firewalls with n rules each, where n is varied from 100 to 2000 in steps of 100. For each value of n, we generated 100 random firewalls. Each generated firewall checks five fields, and the domain of values for each field is the integer interval $[0, 2^{16} - 1]$. For each generated firewall, we used the first 10 rules in the firewall as properties, and applied our deterministic and probabilistic passes in order to verify whether the generated firewall satisfies each of these 10 properties.

The results of the first experiment are summarized in Table I. Note that each line in Table I represents the results of 1000 applications of our deterministic and probabilistic passes to verify whether a randomly generated firewall satisfies a property (which happens to be one of the first 10 rules in

# Rules	Verification Results				Time (sec)
	a(%)	b(%)	c_1 (%)	c_2 (%)	
100	50.9	38.1	0.3	10.7	0.0
200	54.7	34.8	0.6	9.9	0.1
300	53.0	35.9	0.4	10.7	0.1
400	52.0	37.7	0.5	9.8	0.1
500	53.7	35.6	0.6	10.1	0.2
600	54.6	35.1	0.5	9.8	0.2
700	51.0	36.6	0.3	12.1	0.3
800	52.7	34.7	0.7	11.9	0.3
900	53.1	35.2	0.9	10.8	0.4
1000	55.8	34.5	0.4	9.3	0.3
1100	54.6	34.2	0.8	10.4	0.3
1200	52.1	35.8	0.4	11.7	0.4
1300	50.7	38.4	0.2	10.7	0.4
1400	54.2	34.2	0.3	11.3	0.4
1500	54.0	35.0	0.2	10.8	0.4
1600	52.7	36.6	0.9	9.8	0.5
1700	53.4	37.5	1.0	8.1	0.6
1800	54.3	35.3	0.3	10.1	0.5
1900	55.5	34.1	0.2	10.2	0.6
2000	54.2	35.4	0.5	9.9	0.6

TABLE I
RESULTS WHERE THE FIRST 10 RULES IN THE FIREWALL ARE USED AS PROPERTIES

# Rules	Verification Results				Time (sec)
	a(%)	b(%)	c_1 (%)	c_2 (%)	
100	0.2	55.2	0.4	44.2	0.1
200	0.6	56.4	0.0	43.0	0.1
300	0.4	59.7	0.2	39.7	0.2
400	0.1	60.6	0.3	39.0	0.2
500	0.3	56.1	0.3	43.3	0.3
600	1.1	54.7	0.1	44.1	0.4
700	0.3	59.7	0.1	39.9	0.4
800	0.3	55.2	0.5	44.0	0.6
900	0.2	58.7	0.2	40.9	0.6
1000	0.4	57.7	0.6	41.3	0.7
1100	0.3	54.2	0.3	45.2	0.8
1200	0.6	55.6	0.1	43.7	0.8
1300	0.5	57.9	0.3	41.3	0.7
1400	0.4	57.8	0.4	41.4	0.9
1500	0.2	55.5	0.2	44.1	0.9
1600	0.4	55.2	0.1	44.3	1.0
1700	0.7	57.5	0.4	41.4	1.2
1800	0.4	57.3	0.1	42.2	1.1
1900	0.8	55.9	0.4	42.9	1.2
2000	0.6	57.6	0.4	41.4	1.3

TABLE II
RESULTS WHERE THE LAST 10 RULES IN THE FIREWALL ARE USED AS PROPERTIES

the generated firewall). For instance, the first line in Table I indicates that out of 1000 applications,

- 509 produced outcome (a) of the deterministic pass
- 381 produced outcome (b) of the deterministic pass
- 3 produced outcome (c_1) of the probabilistic pass
- 107 produced outcome (c_2) of the probabilistic pass

The average execution time for each application is 0.0 seconds.

B. Experiment 2

The second experiment is similar to the first experiment, except that for each generated firewall, we used the last 10 rules in the firewall as properties, and applied our deterministic and probabilistic passes in order to verify whether the generated firewall satisfies each of these 10 properties.

The results of the second experiment are summarized in Table II.

C. Experiments 3 and 4

In Experiment 3, as in Experiment 1, the properties we verified were obtained from the first 10 rules. However, this time the rules were *flipped* before being used as properties; the predicates of the properties were left unaltered, but the decision was changed to *accept* if the original decision of the rule was *discard*, and vice versa.

Similarly, Experiment 4 used the last 10 rules, like Experiment 2, but this time the rules were flipped. All other details (number of firewalls, number of fields, number of properties) were identical in all four experiments.

The results from our algorithm are stable with regard to the length of the firewalls considered, after $80,000$ trials, and are summarized in Table III. From this table, we can now answer the three questions which we posed at the beginning of this section.

Expt #	Verification results			
	a(%)	b(%)	c_1 (%)	c_2 (%)
1	54.4	36.1	0.4	10.1
2	0.5	54.5	0.3	44.2
3	0.1	76.1	0.0	23.8
4	0.3	67.5	0.1	32.1

TABLE III
SUMMARY OF RESULTS

1) The execution time of our working implementation, to verify that a given firewall satisfies a given property, is about 0.5 seconds.
2) A majority of the properties (between 55% and 80%) are verified using the deterministic pass only.
3) The percentage of times that our verification algorithm produces outcome (c_1) is less than 1.0%.

From Table III, and from our conclusion of the previous section that the probability of error when our verification algorithm produces outcome (c_1) is less than .6%, we conclude that the probability of error our verification algorithm is less than $.006 \times .01 = 6 \times 10^{-5}$.

Besides verifying the efficiency of the algorithm, our experiments produced an unexpected useful result. We observe that, unless a property is proved to be satisfied by the deterministic pass, the chance of the property being proved to be satisfied by the probabilistic pass is extremely low. Thus, the chance of an erroneous result is vanishingly small.

VIII. VERIFICATION OF GENERAL PROPERTIES

So far, we have presented an algorithm to verify whether a given firewall F satisfies a given property P, provided that P has the special form:

$$f_1 \in T_1 \wedge .. \wedge f_d \in T_d \rightarrow < decision >$$

In this form, the predicate of property P is a conjunction of d terms, each term is of the form $(f_j \in T_j)$, and each T_j is an interval taken from the domain of field f_j. In this section, we argue that our algorithms can still be applied to verify whether F satisfies P even if the predicate of P is incomplete or has disjunction and negation operators. For convenience, we carry our argument using the firewall example F_1 in Section II. Recall that firewall F_1 has only three fields f_1, f_2, and f_3, and that the domain of each one of these three fields is the integer interval $[0, 10]$.

A. Incomplete Predicates

Assume that we need to verify whether firewall F_1 satisfies the following property P, whose predicate is incomplete:

$$f_2 \in [5, 8] \rightarrow < decision >$$

In this case, we can use our algorithms to verify whether F_1 satisfies the following equivalent property P', whose predicate is complete:

$$f_1 \in [0, 10] \wedge f_2 \in [5, 8] \wedge f_3 \in [0, 10] \rightarrow < decision >$$

B. Disjunctive Predicates

Assume that we need to verify whether firewall F_1 satisfies the following property P, whose predicate has a disjunction:

$$f_1 \in [2, 7] \vee f_2 \in [5, 8] \rightarrow < decision >$$

In this case, we can use our algorithms to verify whether F_1 satisfies the following two properties P' and P'', whose predicates have no disjunctions:

$$f_1 \in [2, 7] \wedge f_2 \in [0, 10] \wedge f_3 \in [0, 10] \rightarrow < decision >$$
$$f_1 \in [0, 10] \wedge f_2 \in [5, 8] \wedge f_3 \in [0, 10] \rightarrow < decision >$$

If it is concluded that F_1 satisfies both P' and P'', we conclude that F_1 satisfies the original property P. Otherwise, we conclude that F_1 does not satisfy P.

C. Negated Predicates

Assume that we need to verify whether firewall F_1 satisfies the following property P, whose predicate has a negation:

$$\neg f_2 \in [5, 8] \rightarrow < decision >$$

In this case, we can use our algorithms to verify whether F_1 satisfies the following two properties P' and P'', whose predicates have no negations:

$$f_1 \in [0, 10] \wedge f_2 \in [0, 4] \wedge f_3 \in [0, 10] \rightarrow < decision >$$
$$f_1 \in [0, 10] \wedge f_2 \in [9, 10] \wedge f_3 \in [0, 10] \rightarrow < decision >$$

If it is concluded that F_1 satisfies both P' and P'', we conclude that F_1 satisfies the original property P. Otherwise, we conclude that F_1 does not satisfy P.

D. General Predicates

Assume that we need to verify whether firewall F_1 satisfies the following property P, whose predicate has conjunction, disjunction and negation:

$$f_1 \in [2, 7] \wedge \neg(f_2 \in [5, 8] \vee \neg f_3 \in [3, 6]) \rightarrow < decision >$$

In this case, we can use our algorithms to verify whether F_1 satisfies the following two properties P' and P'', whose predicates have conjunction only:

$$f_1 \in [2, 7] \wedge f_2 \in [0, 4] \wedge f_3 \in [3, 6] \rightarrow < decision >$$
$$f_1 \in [2, 7] \wedge f_2 \in [9, 10] \wedge f_3 \in [3, 6] \rightarrow < decision >$$

If it is concluded that F_1 satisfies both P' and P'', we conclude that F_1 satisfies the original property P. Otherwise, we conclude that F_1 does not satisfy P.

IX. Related Work

The problem of ensuring that a given firewall is correct is of great importance. This important problem has been addressed using four approaches: firewall testing, firewall analysis, firewall verification, and firewall design. We briefly discuss these four approaches next.

1) *Firewall Testing:*
To test a given firewall F, one generates many 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 correct. Otherwise, the given firewall F has errors. Different methods of firewall testing differ in how the testing packets are generated. For instance, the test packets can be hand-generated by domain experts to target specific vulnerabilities in the given firewall F, or generated from the formal specifications of the security policy of the given firewall F, as in [11]. A scheme for targeting test packets for better fault coverage is given in [5]. Al-Shaer et al. provide a complete framework to generate targeted packets and obtain good coverage in testing in [2].
2) *Firewall Analysis:*
To analyze a given firewall F, one applies an algorithm to identify (some or all of the) vulnerabilities, conflicts, anomalies, and redundancies in the given firewall F . A systematic method for analyzing firewalls is presented in [15]. The concept of conflicts between rules in a firewall is due to [6] and [1]. A classification of anomalies, as well as algorithms to detect them, may be found in [4] and [3]. (This analysis works for verifying the security policies in IPsec and VPN as well [10].) A framework for understanding the vulnerabilities in a single firewall is outlined in [7], and an analysis of these vulnerabilities presented in [12]. [16] is a quantitative

study of configuration errors for a firewall. An example of an efficient firewall analysis algorithm is given in FIREMAN [17].

3) *Firewall Verification:*
To verify a given firewall F against a given property R, one applies an algorithm (similar to the one discussed in this paper) to verify whether or not F satisfies 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 [15] and [14]. These algorithms are proved to be $O(n^d)$ in [9]. Our paper presents a new, $O(nd)$ algorithm.
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 [8] and [13].

X. Concluding Remarks

We presented in this paper a linear time algorithm for verifying whether a given firewall satisfies a given property. The time complexity of this algorithm is $O(nd)$, where n is the number of rules in the firewall and d is the number of fields that are checked in the firewall or the property. Because verifying whether a given firewall satisfies a given property requires that each conjunct in each rule in the given firewall be compared with the given property, and because the given firewall has nd such conjuncts, the time complexity of any other algorithm for firewall verification can not be less than $O(nd)$, the time complexity of our algorithm.

Our verification algorithm consists of two passes: a deterministic pass followed by a probabilistic pass. The probabilistic pass is executed only when the deterministic pass has the outcome (c), i.e. when the deterministic pass cannot reach a definite conclusion as to whether the given firewall satisfies the given property or not. There are two possible outcomes of the probabilistic pass: (c_1) and (c_2). The (c_1) outcome is that the given firewall satisfies the given property with a high probability, and the (c_2) outcome is that the given firewall does not satisfy the given property. Thus, when our verification algorithm produces outcome (c_1), there is a possibility that this outcome is erroneous. In Section VI, we showed by analysis that the probability of error when our algorithm produces outcome (c_1) is less than $.006$. Then in Section VII, we showed by simulation that the probability of our algorithm producing outcome (c_1) is less than $.01$. This suggests that the probability of our algorithm producing an erroneous result is less than 6×10^{-5} .

This means that if our algorithm is used to verify that a given firewall satisfies $100,000$ given properties, then the results of at most 6 of these properties are erroneous.

The algorithm presented in this paper is limited to verifying a single firewall. However, we can use this algorithm in conjunction with the techniques described in [9] in order to verify enterprise networks that have tens or even hundreds of firewalls.

The task of verifying whether a given firewall satisfies a given property is a fundamental one, and it can be used to perform other tasks such as the task of checking whether a particular rule in a given firewall is redundant. The problem, of whether our linear time verification algorithm can lead to a linear time redundancy checking algorithm, remains open and merits further research.

References

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

[2] 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.

[3] 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.

[4] E. S. Al-Shaer and H. H. Hamed. Discovery of policy anomalies in distributed firewalls. In *INFOCOM*, 2004.

[5] A. El-Atawy, K. Ibrahim, H. Hamed, and E. S. Al-Shaer. Policy segmentation for intelligent firewall testing. *Secure Network Protocols, 2005. (NPSec). 1st IEEE ICNP Workshop on*, pages 67–72, Nov. 2005.

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

[7] 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.

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

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

[10] H. H. Hamed, E. S. Al-Shaer, and W. Marrero. Modeling and verification of ipsec and vpn security policies. In *ICNP*, pages 259–278, 2005.

[11] J. Jürjens and G. Wimmel. Specification-based testing of firewalls. In *PSI '02: Revised Papers from the 4th International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, pages 308–316, London, UK, 2001. Springer-Verlag.

[12] 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.

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

[14] A. X. Liu and M. G. Gouda. Firewall policy queries. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, to appear 2009.

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

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

[17] 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.