Firewall Design: Consistency, Completeness, and Compactness

Mohamed G. Gouda and Xiang-Yang Alex Liu

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188, U.S.A.
{gouda, alex}@cs.utexas.edu

Abstract

A firewall is often placed at the entrance of each private network in the Internet. The function of a firewall is to examine each packet that passes through the entrance and decide whether to accept the packet and allow it to proceed or to discard the packet. A firewall is usually designed as a sequence of rules. To make a decision concerning some packets, the firewall rules are compared, one by one, with the packet until one rule is found to be satisfied by the packet: this rule determines the fate of the packet. In this paper, we present the first ever method for designing the sequence of rules in a firewall to be consistent, complete, and compact. Consistency means that the rules are ordered correctly, completeness means that every packet satisfies at least one rule in the firewall, and compactness means that the firewall has no redundant rules. Our method starts by designing a firewall decision diagram (FDD, for short) whose consistency and completeness can be checked systematically (by an algorithm). We then apply a sequence of five algorithms to this FDD to generate, reduce and simplify the target firewall rules while maintaining the consistency and completeness of the original FDD.

1. Introduction

A firewall is often placed at each entry point of private network in the Internet. The function of this firewall is to provide secure access to and from the private network. In particular, any packet that attempts to enter or leave the private at some entry point is first examined by the firewall located at that point, and depending on the different fields of the packet, the firewall decides either to accept the packet and allow it to proceed in its way, or to discard the packet.

To perform its function, a firewall consists of a sequence of rules; each rule is of the form

$$\langle predicate \rangle \rightarrow \langle decision \rangle$$

where the $\langle predicate \rangle$ is a boolean expression over the different fields of a packet, and the $\langle decision \rangle$ is either "a" (for accept) or "d" (for discard). To reach a decision concerning a packet, the rules in the sequence are examined one by one until the first rule, whose $\langle predicate \rangle$ is satisfied by the packet fields, is found. The $\langle decision \rangle$ of this rule is applied to the packet.

Designing the sequence of rules for a firewall is not any easy task. In fact, the sequence of rules for any firewall needs to be consistent, complete, and compact as we illustrated by the next (admittedly simple) example.

Example: Figure 1 shows a private network that contains a mail server s and a host h. The private network is connected to the firewall via interface 1, whereas the rest of the Internet is connected to the firewall via interface 0. The rest of the Internet has a malicious host m.

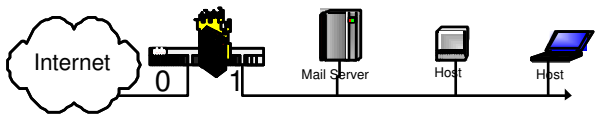


Figure 1. A firewall in a simple network

In this example, we assume that each packet has five fields named as follows:

I is the interface on which the packet reaches the firewall

S is the original source of the packet

D is the ultimate destination of the packet

P is the transport protocol of the packet

T is the destination port of the packet

The firewall in this example consists of the following sequence

$$
\begin{aligned}
(\ & I = 0 \wedge S = any \wedge D = s \wedge P = tcp \wedge T = 25 \rightarrow a, \\
& I = 0 \wedge S = any \wedge D = s \wedge P = any \wedge T = any \rightarrow d, \\
& I = 0 \wedge S = m \wedge D = any \wedge P = any \wedge T = any \rightarrow d, \\
& I = 1 \wedge S = h \wedge D = any \wedge P = any \wedge T = any \rightarrow a, \\
& I = 1 \wedge S = any \wedge D = any \wedge P = any \wedge T = any \rightarrow a)
\end{aligned}
$$

We refer to these five rules as r_0 through r_4, respectively. Rule r_0 accepts each incoming SMTP packet whose ultimate destination is a mail server s. Rule r_1 discards each incoming non-SMTP packet whose ultimate destination is s. Rule r_2 discards each incoming packet whose original source is a malicious host m. Rule r_3 accepts each outgoing packet whose original source is a host h. Rule r_4 accepts each outgoing packet. Next, we argue that this sequence of five rules suffers from three types of errors: consistency errors, completeness errors, and compactness errors.

The two rules r_0 and r_2 are conflicting because there are packets whose fields satisfy the predicates of both r_0 and r_2 (for example, a packet where $I = 0$, $S = m$, $D = s$, $P = tcp$, and $T = 25$ satisfies the predicates of both r_0 and r_2) and these two rules have different decisions. Therefore, the relative order of these two rules with respect to one another in the sequence of rules becomes very critical. For example, by placing rule r_2 behind rule r_0 in the above rule sequence, the firewall accepts all incoming SMTP packets even those that originated at the malicious host m. This relative order of rules r_0 and r_2 is likely a *consistency error*. This error can be corrected by placing rule r_2 at the beginning of the sequence of rules ahead of rule r_1. This correction will cause the firewall to discard all incoming packets, including SMTP packets, that originated at the malicious host m.

The second error in the above rule sequence is that any packet where $I = 0$, $S \neq m$, and $D \neq s$ does not satisfy the predicate of any of the five rules r_0 through r_5. We refer to such an error as a *completeness error*. This error can be corrected by adding the following new rule immediately before rule r_3

$$I = 0 \wedge S = any \wedge D = any \wedge P = any \wedge T = any \rightarrow a$$

The third error in the above rule sequence is that rule r_3 is redundant; i.e., this rule can be removed without affecting the set of all packets accepted by the rule sequence and without affecting the set of all packets discarded by the rule sequence. We refer to such an error as a *compactness error*. This error can be corrected by removing rule r_3 from the above rule sequence.

After we preform these corrections, we end up with the following sequence of rules.

$$
\begin{aligned}
(\ & I = 0 \wedge S = m \wedge D = any \wedge P = any \wedge T = any \rightarrow d, \\
& I = 0 \wedge S = any \wedge D = s \wedge P = tcp \wedge T = 25 \rightarrow a, \\
& I = 0 \wedge S = any \wedge D = s \wedge P = any \wedge T = any \rightarrow d, \\
& I = 0 \wedge S = any \wedge D = any \wedge P = any \wedge T = any \rightarrow a, \\
& I = 1 \wedge S = any \wedge D = any \wedge P = any \wedge T = any \rightarrow a)
\end{aligned}
$$

In this paper, we present a method for designing the sequence of rules of a firewall to be consistent, complete, and compact. According to this method, a firewall designer starts by specifying what we call a firewall decision diagram whose consistency and completeness can be checked systematically (by an algorithm). Then, the designer applies a sequence of five algorithm to this firewall decision diagram to generate a compact sequence of firewall rules while maintaining the consistency and completeness of the original diagram.

2. Related Work

Design errors in existing firewalls have been reported in [3]. Yet, most of the research in the area of firewalls and packet classifiers was not dedicated to the problem of how to design correct firewalls. Rather, most of the research in this area was dedicated to developing efficient data structures that can speed up the checking of firewall rules when a new packet reaches a firewall. Examples of such data structures are the trie data structures in [12], area-based quadtrees [6], fat inverted segment trees [8]. A good survey of these data structures is presented in [9].

Another research direction in the area of firewall design has been dedicated to the development of high-level specification languages that can be used in specifying firewall rules. Examples of such languages are the simple model definition language in [3], the Lisp-like language in [10], the declarative predicate language in [4], and the high level firewall language in [1]. However, none of these specification languages is able to simplify the task of ensuring consistency, completeness, and compactness of the firewalls being specified.

Perhaps the research direction that is closest to the spirit of the current papers is reported in [11], [7], [2]. This research direction is dedicated to detecting every pair of conflicting rules in a firewall. Each detected pair of conflicting rules is then presented to the firewall designer who decides whether the two rules need to be swapped or a new rule need to be added. However, these methods do not deal with the completeness and compactness errors of a firewall.

According to the firewall design method in the current paper, firewalls are first specified as firewall decision diagrams. These decision diagrams are similar, but not identical, to other types of decision diagrams such as the Binary Decision Diagrams in [5] and the Interval Decision Diagrams in [13].

3. Firewall Decision Diagrams (FDDs)

A *field* F_i is a variable whose value is taken from a predefined interval of nonnegative integers, called the *domain* of F_i and denoted by $D(F_i)$.

A *packet* over the fields $F_0, \cdots, F_{n-1}$ is an n-tuple $(p_0, \cdots, p_{n-1})$ where each p_i is taken from the domain $D(F_i)$ of the corresponding field F_i.

A *firewall decision diagram* f (or FDD f, for short) over the fields $F_0, \cdots, F_{n-1}$ is an acyclic and directed graph that satisfies the following five conditions:

1. f has exactly one node that has no incoming edges, called the *root* of f, and has two or more nodes that have no outgoing edges, called the *terminal nodes* of f.
2. Each nonterminal node v in f is labelled with a field, denoted by $F(v)$, taken from the set of fields $F_0, \cdots, F_{n-1}$. Each terminal node v in f is labelled with a decision that is either accept (or "a" for short) or discard (or "d" for short).
3. A directed path from the root to a terminal node in f is called a *decision path*. No two nodes on a decision path in f have the same label.
4. Each edge e, that is outgoing of a node v in f, is labelled with an integer set $I(e)$, where $I(e)$ is a subset of the domain of field $F(v)$.
5. Let v be any terminal node in f. The set $E(v)$ of all outgoing edges of node v satisfies the following two conditions:
   (a) *Consistency*: For any distinct e_i and e_j in $E(v)$,
   $$I(e_i) \cap I(e_j) = \emptyset$$
   (b) *Completeness*: $\bigcup_{e \in E(v)} I(e) = D(F(v))$
   where $\emptyset$ is the empty set and $D(F(v))$ is the domain of the field $F(v)$.

Figure 2 shows an FDD over two fields F_0 and F_1. The domain of each field is the interval $[0, 9]$. Each edge in this FDD is labelled with a set of integers that is represented by one or more non-overlapping intervals (that cover the set of integers). For example, one outgoing edge of the root is labelled with the two intervals $[0, 3]$ and $[8, 9]$ that represent the set $\{0, 1, 2, 3, 8, 9\}$.

Let f be an FDD over the fields $F_0, \cdots, F_{n-1}$. A decision path in f is denoted $(v_0 e_0 \cdots v_{k-1} e_{k-1} v_k)$ where v_0 is the root node in f, v_k is a terminal node in f, and each e_i is a directed edge from node v_i to node v_{i+1} in f.

Each decision path $(v_0 e_0 \cdots v_{k-1} e_{k-1} v_k)$ in an FDD f over the packet fields $F_0, \cdots, F_{n-1}$ can be represented as a rule of the form:

$$F_0 \in S_0 \wedge \cdots \wedge F_{n-1} \in S_{n-1} \ \rightarrow \ \langle decision \rangle$$

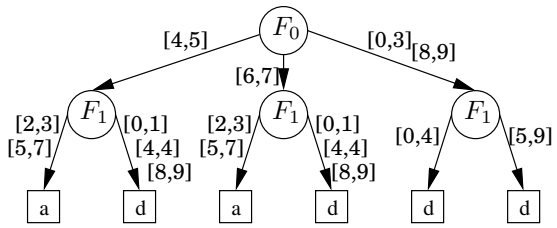


Figure 2. An FDD

where $\langle decision \rangle$ is the label of the terminal node v_k in the decision path and each field F_i satisfies one of the following two conditions:

1. No node in the decision path is labelled with field F_i and S_i is the domain of F_i.
2. There is one node v_j in the decision path that is labelled with field F_i and S_i is the label of edge e_j in the decision path.

An FDD f over the fields $F_0, \cdots, F_{n-1}$ can be represented by a sequence of rules, each of them is of the form

$$F_0 \in S_0 \wedge \cdots \wedge F_{n-1} \in S_{n-1} \ \rightarrow \ \langle decision \rangle$$

such that the following two conditions hold. First, each rule in the sequence represents a distinct decision path in f. Second, each decision path in f is represented by a distinct rule in the sequence. Note that the order of the rules in the sequence is immaterial.

We refer to a sequence of rules that represents an FDD f as a *firewall of* f.

A packet $(p_0, \cdots, p_{n-1})$ over the fields $F_0, \cdots, F_{n-1}$ is said to be *accepted by* an FDD f over the same fields iff a firewall of f has a rule

$$F_0 \in S_0 \wedge \cdots \wedge F_{n-1} \in S_{n-1} \ \rightarrow \ accept$$

such that the condition $p_0 \in S_0 \wedge \cdots \wedge p_{n-1} \in S_{n-1}$ holds.

Similarly, a packet $(p_0, \cdots, p_{n-1})$ over the fields $F_0, \cdots, F_{n-1}$ is said to be *discarded by* an FDD f over the same fields iff a firewall of f has a rule

$$F_0 \in S_0 \wedge \cdots \wedge F_{n-1} \in S_{n-1} \ \rightarrow \ discard$$

such that the condition $p_0 \in S_0 \wedge \cdots \wedge p_{n-1} \in S_{n-1}$ holds.

Let Σ denote the set of all packets over the fields $F_0, \cdots, F_{n-1}$, and let f be an FDD over the same fields. The *accept set* of f, denoted $f.accept$, is the subset of Σ that contains all the packets accepted by f. Similarly, the *discard set* of f, denoted $f.discard$, is the subset of Σ that contains all the packets discarded by f.

Theorem 1 (Theorem of FDDs) For any FDD f over the fields $F_0, \cdots, F_{n-1}$,

1. $f.accept \cap f.discard = \emptyset$, and
2. $f.accept \cup f.discard = \Sigma$

where $\emptyset$ is the empty set and Σ is the set of all packets over the fields $F_0, \cdots, F_{n-1}$. □

Two FDDs f and f' over the same fields are said to be *equivalent* iff they have identical accepts sets and identical discard sets, i.e.,

$$f.accept = f'.accept, \text{ and}$$
$$f.discard = f'.discard.$$

4. Reduction of FDDs

As discussed in the precious section, the number of rules in a firewall of an FDD f equals the number of decision paths in f. Thus, it is advantageous to reduce the number of decision paths in an FDD without changing its semantics, i.e., without changing its accept and discard sets. The procedure for reducing the number of decision paths in an FDD without changing its accept and discard sets is called a *reduction* of this FDD. This procedure is discussed in this section. But before we introduce the concept of a reduced FDD, we need to introduce the concept of isomorphic nodes in an FDD.

Two nodes v_0 and v_1 in an FDD f are called *isomorphic* in f iff v_0 and v_1 satisfy one of the following two conditions:

1. Both v_0 and v_1 are terminal nodes with identical labels in f.
2. Both v_0 and v_1 are nonterminal nodes and there is a one-to-one correspondence between the outgoing edges of v_0 and the outgoing edges of v_1 such that every pair of corresponding edges have identical labels and are incoming of the same node in f.

An FDD f is called *reduced* iff it satisfies the following three conditions:

1. f has no node with exactly one outgoing edge.
2. f has no two edges that are outgoing of one node and are incoming of another node.
3. f has no two distinct isomorphic nodes.

The reduction procedure of FDDs is presented next.

Algorithm 1: (Reduction of FDDs)
input : an FDD f
output: a reduced FDD that is equivalent to f
steps:

Repeatedly apply the following three reductions to f until none of them can be applied any further.

1. If f has a node v_0 with only one outgoing edge e and if e is incoming of another node v_1, then remove v_0 and e from f and make the incoming edges of v_0 incoming of v_1.

2. If f has two edges e_0 and e_1 that are outgoing of node v_0 and incoming of node v_1, then remove e_0 and make the label of e_1 be the integer set $I(e_0) \cup I(e_1)$, where $I(e_0)$ and $I(e_1)$ are the integer sets that labelled edges e_0 and e_1 respectively.

3. If f has two isomorphic nodes v_0 and v_1, then remove node v_1 and its outgoing edges, and make the incoming edges of v_0 incoming of v_1.

Applying Algorithm 1 to the FDD in Figure 2 yields the reduced FDD in Figure 3. Note that a firewall of the FDD in Figure 2 consists of six rules, whereas a firewall of the FDD in Figure 3 consists of three rules.

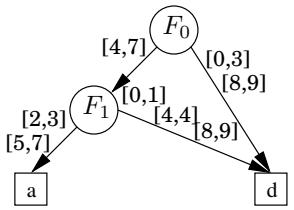


Figure 3. A reduced FDD

5. Marking of FDDs

In section 8, we describe an algorithm for replacing each rule in the firewall of a reduced FDD f by a sequence of "simple" rules. The total number of the resulting simple rules in the firewall equals the "degree" of a "marked version" of f. Next, we define what we mean by a marked version of an FDD and its degree.

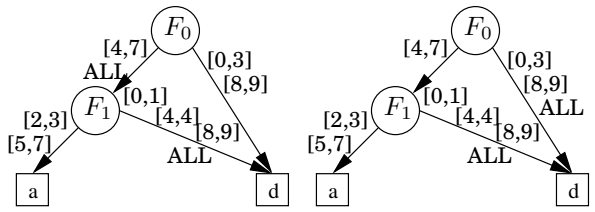
A *marked version* f' of a reduced FDD f is the same as f except that exactly one outgoing edge of each nonterminal node in f' is marked "ALL". We adopt the convention:

$$f.accept = f'.accept, \text{ and}$$
$$f.discard = f'.discard.$$

We sometimes refer to f' as a *marked FDD*.

Figure 4 shows two marked versions f' and f'' of the reduced FDD in Figure 3. In f', the edge labelled $[4, 7]$ and the edge labelled $[0, 1][4, 4][8, 9]$ are both marked ALL. In f'', the edge labelled $[0, 3][8, 9]$ and the edge labelled $[0, 1][4, 4][8, 9]$ are both both marked ALL.

The *degree of a set of integers* S, denoted $deg(S)$, is the smallest number of non-overlapping integer intervals that cover S. For example, the degree of the set



(a) f' (b) f''

Figure 4. Two marked FDDs

$\{0, 1, 2, 4, 7, 8, 9\}$ is 3 because this set is covered by the three integer intervals $[0, 2]$, $[4, 4]$ and $[7, 9]$.

The *degree of an edge* e in a marked FDD, denoted $deg(e)$, is defined as follows. If e is marked ALL, then $deg(e) = 1$. If e is not marked ALL, then $deg(e) = deg(S)$ where S is the set of integers that labels edge e.

The *degree of a node* v in a marked FDD, denoted $deg(v)$, is defined recursively as follows. If v is a terminal node, then $deg(v) = 1$. If v is a nonterminal node with k outgoing edges $e_0, \cdots, e_{k-1}$ that are incoming of nodes $v_0, \cdots, v_{k-1}$ respectively, then

$$deg(v) = \sum_{i=0}^{k-1} deg(e_i) \times deg(v_i)$$

The *degree of a marked FDD* f, denoted $deg(f)$, equals the degree of the root node of f.

For example, $deg(f') = 5$ where f' is the marked FDD in Figure 4(a), and $deg(f'') = 4$ where f'' is the marked FDD in Figure 4(b).

From the above examples, a reduced FDD may have many marked versions, and each marked version may have a different degree. As mentioned earlier, the number of simple rules in the firewall of a marked FDD equals the degree of the marked FDD. Thus, it is advantageous, when generating a marked version of a reduced firewall, to generate the marked version with the smallest possible degree. This is achieved by the following algorithm.

Algorithm 2: (Marking of FDDs)
input : a reduced FDD f
output: a marked version f' of f such that for every marked version f'' of f, $deg(f') \leq deg(f'')$
steps:

1. Compute the degree of each terminal node v in f as follows:
   $$deg(v) = 1$$
2. **while** f has a node v whose degree has not yet been computed and v has k outgoing edges $e_0, \cdots, e_{k-1}$ that are incoming of the nodes $v_0, \cdots, v_{k-1}$, respectively, whose degrees have already been computed **do**
   (1) Find an outgoing edge e_j of v whose quantity
   $$(deg(e_j) - 1) \times deg(v_j)$$
   is larger than or equal to the corresponding quantity of every other outgoing edge of v.
   (2) Mark edge e_j with "ALL".
   (3) Compute the degree of v as follows:
   $$deg(v) = \sum_{i=0}^{k-1} (deg(e_i) \times deg(v_i))$$

end

If Algorithm 2 is applied to the reduced FDD in Figure 3, we obtain the marked FDD in Figure 4(b).

6. Firewall Generation

In this section, we present an algorithm, Algorithm 3, for generating a firewall of a marked FDD f, which is generated by Algorithm 2 in section 5. The generated firewall is a sequence of rules where each rule corresponds to a decision path in the marked FDD f. Algorithm 3 computes for each rule in the generated firewall a binary number, called *rank* of the rule, and two predicates, called *exhibited* and *original predicates* of the rule. The rule ranks are used to order the computed rules in the generated firewall. The exhibited and original predicates of the rules are used in the next section to make the generated firewall "compact".

A *firewall* r over the fields $F_0, \cdots, F_{n-1}$ is a sequence of rules $r_0, \cdots, r_{m-1}$ where each rule is of the form

$$F_0 \in S_0 \wedge \cdots \wedge F_{n-1} \in S_{n-1} \ \rightarrow \ \langle decision \rangle$$

Each S_i is either the mark ALL or a nonempty set of integers taken from the domain of field F_i (which is an interval of consecutive nonnegative integers). The $\langle decision \rangle$ is either a (for accept) or d (for discard). The last rule, r_{m-1}, in firewall r is of the form:

$$F_0 \in S_0 \wedge \cdots \wedge F_{n-1} \in S_{n-1} \ \rightarrow \ \langle decision \rangle$$

where each S_i is either the mark ALL or the entire domain of field F_i.

A packet $(p_0, \cdots, p_{n-1})$ over the fields $F_0, \cdots, F_{n-1}$ is said to *match* a rule r_i in a firewall over the same fields iff rule r_i is of the form

$$F_0 \in S_0 \wedge \cdots \wedge F_{n-1} \in S_{n-1} \ \rightarrow \ \langle decision \rangle$$

and the predicate $(p_0 \in S_0 \wedge \cdots \wedge p_{n-1} \in S_{n-1})$ holds.

A packet over the fields $F_0, \cdots, F_{n-1}$ is said to be *accepted* by a firewall r over the same fields iff r has a rule r_i that satisfies the following three conditions:

1. The packet matches r_i.
2. The packet does not match any rule that precedes r_i.
3. The $\langle decision \rangle$ of r_i is a.

Similarly, a packet over the fields $F_0, \cdots, F_{n-1}$ is said to be *discarded* by a firewall r over the same fields iff r has a rule r_i that satisfies the following three conditions:

1. The packet matches r_i.
2. The packet does not match any rule that precedes r_i.
3. The $\langle decision \rangle$ of r_i is d.

Let r be a firewall over the fields $F_0, \cdots, F_{n-1}$. The set of all packets accepted by r is denoted $r.accept$, and the set of all packets discarded by r is denoted $r.discard$. The next theorem follows from these definitions.

Theorem 2 (Theorem of Firewalls) For any firewall r over the fields $F_0, \cdots, F_{n-1}$,

1. $r.accept \cap r.discard = \emptyset$, and
2. $r.accept \cup r.discard = \Sigma$

where $\emptyset$ is the empty set and Σ is the set of all packets over the fields $F_0, \cdots, F_{n-1}$. □

Algorithm 3: (Firewall Generation)

input : a marked FDD f over the fields $F_0, \cdots, F_{n-1}$ and assume that along each directed path in f, if a field F_i appears before field F_j then $i < j$.

output: a firewall r over the same fields such that
$r.accept = f.accept$, and
$r.accept = f.accept$
and for each rule r_i in r, the algorithm computes a binary number of n bits, called the rank of r_i, and two predicates, called the exhibited and original predicates of r_i.

steps:

1. For each decision path in f, compute a rule r_i, its rank, its exhibited predicate ep_i and its original predicate op_i as follows:
   $F_0 \in S_0 \wedge \cdots \wedge F_{n-1} \in S_{n-1} \rightarrow \langle decision \rangle$
   rank = $b_0 \cdots b_{n-1}$
   $ep_i = (F_0 \in S_0 \wedge \cdots \wedge F_{n-1} \in S_{n-1})$
   $op_i = (F_0 \in T_0 \wedge \cdots \wedge F_{n-1} \in T_{n-1})$
   where each b_i, S_i, and T_i is computed according to the following three cases:
   Case 1:(*The decision path has no nodes labelled* F_i)
   $b_i = 0$
   S_i =the domain $[a_i, b_i]$ of F_i
   T_i =the domain $[a_i, b_i]$ of F_i
   Case 2:(*The decision path has a node labelled* F_i, *and its outgoing edge* e *has no mark*)
   $b_i = 0$
   S_i =the integer set that labels e
   T_i =the integer set that labels e
   Case 3:(*The decision path has a node labelled* F_i, *and its outgoing edge* e *has an ALL mark*)
   $b_i = 1$
   S_i =ALL
   T_i =the integer set that labels e
2. After computing all the rules and their ranks, order the rules in r in an ascending order of their ranks.

As an example, if Algorithm 3 is applied to the marked FDD in Figure 4(b), we obtain the firewall in Figure 5. Associated with each of the three rules in this firewall are a rank, and an exhibited and original predicates. In particular, associated with the first rules are

rank =00,
exhibited predicate =$(F_0 \in [4,7] \wedge F_1 \in [2,3] \cup [5,7])$,
original predicate =exhibited predicate.

Associated with the second rule are:

rank =01,
exhibited predicate =$(F_0 \in [4,7] \wedge F_1 \in ALL)$,
original predicate =$(F_0 \in [4,7] \wedge F_1 \in [0,1] \cup [4,4] \cup [8,9])$.

Associated with the third rule are:

rank =10,
exhibited predicate =$(F_0 \in ALL \wedge F_1 \in [0,9])$,
original predicate =$(F_0 \in [0,3] \cup [8,9] \wedge F_1 \in [0,9])$.

Note that the three rules are placed in the firewall in ascending order of their ranks.

$$
\begin{array}{rl}
r = (& F_0 \in [4,7] \wedge F_1 \in [2,3] \cup [5,7] \rightarrow a, \\
& F_0 \in [4,7] \wedge F_1 \in ALL \quad\quad \rightarrow d, \\
& F_0 \in ALL \wedge F_1 \in [0,9] \quad\quad \rightarrow d, \\
)
\end{array}
$$

Figure 5. A generated firewall

7. Firewall Compactness

Firewalls that are generated by Algorithm 3 in the last section can have *redundant rules*, i.e., rules that can be removed from their firewalls without affecting the accept or discard sets of these firewalls. For example, the second rule in firewall r in Figure 5 is redundant. Thus, removing this rule from r yields the equivalent in Figure 6. The two firewalls are equivalent since

$r'.accept = r.accept$, and
$r'.accept = r.accept$

A firewall is called *compact* iff it has no redundant rules. It is straightforward to argue that the firewall in Figure 6 is compact.

In this section, we present an algorithm, Algorithm 4, for detecting and removing all redundant rules from the firewalls generated by Algorithm 3. Algorithm 4 is based on the following theorem.

r' = ($F_0 \in [4,7] \wedge F_1 \in [2,3] \cup [5,7] \rightarrow\ a$,
$F_0 \in ALL \wedge F_1 \in [0,9] \rightarrow\ d$,
)

Figure 6. A compact firewall

Theorem 3 (Redundancy of Firewall Rules) Let $(r_0, \cdots, r_{m-1})$ be a firewall over the fields $F_0, \cdots, F_{n-1}$ generated by Algorithm 3 in the last section. A rule r_i in this firewall, $i < m-1$, is redundant iff for each j, $i < j \leq m-1$, at least one of the following two conditions holds:

1. $\langle decision \rangle$ of $r_j = \langle decision \rangle$ of r_i.
2. No packet over the fields $F_0, \cdots, F_{n-1}$ satisfies the predicate

$$r_i.op \wedge (\neg r_{i+1}.ep \wedge \cdots \wedge \neg r_{j-1}.ep) \wedge r_j.ep$$

where $r_i.op$ denotes the original predicate of r_i and $r_j.ep$ denotes the exhibited predicate of r_j.

□

Algorithm 4: (Firewall Compaction)
input : a firewall r with m rules $(r_0, \cdots, r_{m-1})$ over the fields $F_0, \cdots, F_{n-1}$ generated by Algorithm 3
output: a compact firewall r' such that
$r'.accept = r.accept$, and
$r'.accept = r.accept$
variables
i : $0..m-2$;
j : $0..m$;
$redundant$: **array** $[0..m-1]$ **of boolean**;
np : $name\ of\ a\ predicate$;
steps:
1. $redundant[m-1] := false$;
2. **for** $i = m-2$ **to** 0 **do**
   $j := i+1$;
   let $r_i.op$ **be named** np;
   $redundant[i] :=$ **true**
   while $redundant[i] \wedge j \leq m-1$ **do**
   if $redundant[j]$
   then $j := j+1$;
   else if ($\langle decision \rangle\ of\ r_i = \langle decision \rangle\ of\ r_j$)
   $\vee$(*no packet over the fields* $F_0, \cdots,$
   F_{n-1} *satisfies* $np \wedge r_j.ep$)
   then let $np \wedge \neg r_j.ep$ **be named** np;
   $j := j+1$;
   else $redundant[i] := false$;
3. Remove from r every rule r_i where $redundant[i] := true$.

If we apply Algorithm 4 to the firewall in Figure 5, we obtain the compact firewall in Figure 6.

8. Firewall Simplification

A firewall rule of the form

$$F_0 \in S_0 \wedge \cdots \wedge F_{n-1} \in S_{n-1} \rightarrow \langle decision \rangle$$

is called *simple* iff every S_i in the rule is either the ALL mark or an interval of consecutive nonnegative integers. A firewall is called *simple* off all its rules are simple.

The following algorithm can be used to simplify any firewall generated by Algorithm 3 or Algorithm 4.

Algorithm 5: (Firewall Simplification)
input : a firewall r generated by Algorithm 3 or Algorithm 4
output: a simple firewall r' such that
$r'.accept = r.accept$, and
$r'.accept = r.accept$
steps:
while r has a rule of the form
$F_0 \in S_0 \wedge \cdots \wedge F_i \in S \cup [a,b] \wedge \cdots \wedge F_{n-1} \in S_{n-1} \rightarrow \langle decision \rangle$
where S is a nonempty set of nonnegative integers that has neither $a-1$ nor $b+1$
do
replace this rule by two consecutive rules of the form:
$F_0 \in S_0 \wedge \cdots \wedge F_i \in S \wedge \cdots \wedge F_{n-1} \in S_{n-1} \rightarrow \langle decision \rangle$,
$F_0 \in S_0 \wedge \cdots \wedge F_i \in [a,b] \wedge \cdots \wedge F_{n-1} \in S_{n-1} \rightarrow \langle decision \rangle$
end

If we apply Algorithm 5 to the compact firewall r' in Figure 6, we obtain the simple firewall r'' in Figure 7. Note

r = ($F_0 \in [4,7] \wedge F_1 \in [2,3] \rightarrow\ a$,
$F_0 \in [4,7] \wedge F_1 \in [5,7] \rightarrow\ d$,
$F_0 \in ALL \wedge F_1 \in [0,9] \rightarrow\ d$,
)

Figure 7. A simple firewall

that our firewall running example, illustrated in the Figure 2 through 7, started by the FDD in Figure 2. If we directly generate and simplified our firewall from this FDD, ignoring Algorithm 1, 2, and 4, then we would have ended up with a firewall consists of 14 simple rules. Thus the role of Algorithms 1, 2, and 4 is to reduce the number of simple rules in the final firewall from 14 to mere 3. A big saving!

9. Summary of Firewall Design

It is useful to summarize our firewall design method in this section, Figure 8 shows the different steps of our fire-

wall design method.

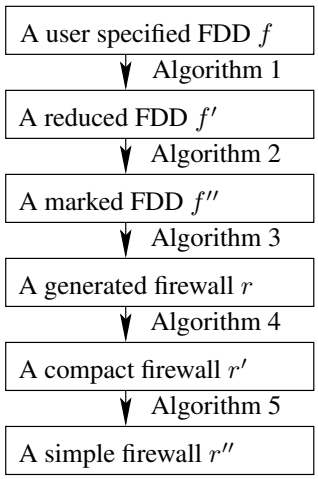


Figure 8. Steps of our firewall design method

The method starts by some user specifying an FDD f. The consistency and completeness properties of f can be verified systematically, possibly using a computer program. Although f guarantees that the final firewall is both consistent and complete, f should not be used to directly generate and simplify this final firewall (otherwise, the number of simple rules in the final firewall would be very large). Instead, f is first reduced (using Algorithm 1), and some of its edges are marked with the ALL mark (using Algorithm 2), then the firewall is generated from the marked FDD (using Algorithm 3).

Note that although marking some of the edges in an FDD introduces conflicts into the FDD, the marking algorithm, Algorithm 3, maintains the consistency and completeness conditions of the original FDD.

Unfortunately, the generated firewall can still have some redundant rules (even though this firewall is generated after applying the reduction algorithm, Algorithm 1, go get rid of many redundant rules). Thus, Algorithm 4 is used to detect and remove all the remaining redundant rules from the generated firewall.

Finally, Algorithm 5 is used to simplify the rules in the generated firewall. Note that the marking algorithm, Algorithm 2, guarantees that the number of simple rules in the generated firewall is kept to a minimum.

10. Concluding Remarks

Our contribution in this paper is two-fold. First, we proposed to use firewall decision diagrams to specify firewalls at the early stage of firewall design. The main advantages of these diagrams is that their consistency and completeness can be checked systematically. Second, we developed a sequence of five algorithms that can be applied to a firewall decision diagram to generate a compact sequence of firewall rules while maintaining the consistency and completeness of the original firewall diagram.

In our firewall design method, we assume that to each encountered packet, a firewall assigns one of two decisions: accept or discard. Nevertheless, this design method can be easily extended to allow a firewall to select one of any number decisions. For example, the extended method can be used to design a firewall that assigns to each encountered packet a decision taken from the following four decisions: accept, accept-and-log, discard, and discard-and-log.

The design method discussed in this paper is not restricted to designing firewalls. Rather, this method can also be applied to the design of general packet classifiers.

References

[1] High level firewall language, http://www.hlfl.org/.

[2] F. Baboescu and G. Varghese. Fast and scalable conflict detection for packet classifiers. In *Proc. of the 10th IEEE International Conference on Network Protocols*, 2002.

[3] Y. Bartal, A. J. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. In *Proc. of IEEE Symp. on Security and Privacy*, pages 17–31, 1999.

[4] A. Begel, S. McCanne, and S. L. Graham. BPF+: Exploiting global data-flow optimization in a generalized packet filter architecture. In *Proc. of ACM SIGCOMM '99*, 1999.

[5] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Computers*, 35(8):677–691, 1986.

[6] M. M. Buddhikot, S. Suri, and M. Waldvogel. Space decomposition techniques for fast Layer-4 switching. In *Proc. of PHSN*, Aug. 1999.

[7] D. Eppstein and S. Muthukrishnan. Internet packet filter management and rectangle geometry. In *Symp. on Discrete Algorithms*, pages 827–835, 2001.

[8] A. Feldmann and S. Muthukrishnan. Tradeoffs for packet classification. In *Proc. of 19th IEEE INFOCOM*, Mar. 2000.

[9] P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, 15(2):24–32, 2001.

[10] J. D. Guttman. Filtering postures: Local enforcement for global policies. In *Proc. of IEEE Symp. on Security and Privacy*, pages 120–129, 1997.

[11] A. Hari, S. Suri, and G. M. Parulkar. Detecting and resolving packet filter conflicts. In *Proc. of IEEE Infocom*, pages 1203–1212, 2000.

[12] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. In *Proc. of ACM SIGCOMM*, pages 191–202, 1998.

[13] K. Strehl and L. Thiele. Interval diagrams for efficient symbolic verification of process networks. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 19(8):939–956, 2000.