

# Complete Redundancy Detection in Firewalls

Alex X. Liu\* and Mohamed G. Gouda

Department of Computer Sciences,  
The University of Texas at Austin,  
Austin, Texas 78712-0233, USA  
{alex, gouda}@cs.utexas.edu

**Abstract.** Firewalls are safety-critical systems that secure most private networks. The function of a firewall is to examine each incoming and outgoing packet and decide whether to accept or to discard the packet. This decision is made according to a sequence of rules, where some rules may be redundant. Redundant rules significantly degrade the performance of firewalls. Previous work detects only two special types of redundant rules. In this paper, we solve the problem of how to detect all redundant rules. First, we give a necessary and sufficient condition for identifying all redundant rules. Based on this condition, we categorize redundant rules into upward redundant rules and downward redundant rules. Second, we present methods for detecting the two types of redundant rules respectively. Our methods make use of a tree representation of firewalls, which is called firewall decision trees.

**Keywords:** Firewall, Redundant Rules, Network Security.

## 1 Introduction

### 1.1 Firewall Basics

Serving as the first line of defense against malicious attacks and unauthorized traffic, firewalls are crucial elements in securing the private networks of most businesses, institutions, and even home networks. A firewall is placed at the point of entry between a private network and the outside Internet so that all incoming and outgoing packets have to pass through it. A packet can be viewed as a tuple with a finite number of fields; examples of these fields are source/destination IP address, source/destination port number, and protocol type. A firewall maps each incoming and outgoing packet to a decision according to its configuration. A firewall configuration defines which packets are legitimate and which are illegitimate by a sequence of rules. Each rule in a firewall configuration is of the form

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

The  $\langle predicate \rangle$  in a rule is a boolean expression over some packet fields and the physical network interface on which a packet arrives. The  $\langle decision \rangle$  of a rule can

---

\* Corresponding author.

be *accept*, or *discard*, or a combination of one of these decisions with other options such as a logging option. For simplicity, we assume that the *decision* in a rule is either *accept* or *discard*. Since the focus of this paper is firewall configuration, later we use “firewall” to mean “firewall configuration” if not otherwise specified.

A packet *matches* a rule if and only if (*iff*) the packet satisfies the predicate of the rule. The predicate of the last rule in a firewall is usually a tautology to ensure that every packet has at least one matching rule in the firewall. Firewall rules often conflict. Two rules in a firewall *conflict* iff they not only overlap but also have different decisions. Two rules overlap iff there is at least one packet that can match both rules. Due to conflicts among rules, a packet may match more than one rule in a firewall, and the rules that a packet matches may have different decisions. To resolve conflicts among rules, for each incoming or outgoing packet, a firewall maps it to the decision of the first (i.e., highest priority) rule that the packet matches.

## 1.2 Redundant Rules

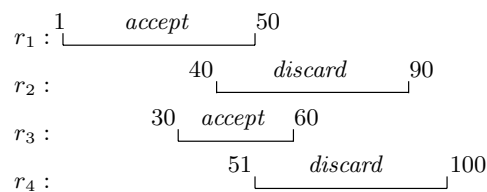
Firewalls often have redundant rules. A rule in a firewall is redundant iff removing the rule does not change the function of the firewall, i.e., does not change the decision of the firewall for every packet. For example, consider the firewall in Figure 1, whose geometric representation is in Figure 2. This firewall consists of four rules  $r_1$  through  $r_4$ . The domain of field  $F_1$  is  $[1, 100]$ .

We have the following two observations concerning the redundant rules in the firewall in Figure 1.

1. Rule  $r_3$  is redundant. This is because the first matching rule for all packets where  $F_1 \in [30, 50]$  is  $r_1$ , and the first matching rule for all packets where  $F_1 \in [51, 60]$  is  $r_2$ . Therefore, there are no packets whose first matching

$$\begin{aligned} r_1 : F_1 \in [1, 50] &\rightarrow \textit{accept} \\ r_2 : F_1 \in [40, 90] &\rightarrow \textit{discard} \\ r_3 : F_1 \in [30, 60] &\rightarrow \textit{accept} \\ r_4 : F_1 \in [51, 100] &\rightarrow \textit{discard} \end{aligned}$$

**Fig. 1.** A simple firewall



**Fig. 2.** Geometric representation of the firewall in Figure 1

rule is  $r_3$ . We call  $r_3$  an upward redundant rule. A rule  $r$  in a firewall is *upward redundant* iff there are no packets whose first matching rule is  $r$ . Geometrically, a rule is upward redundant in a firewall iff the rule is overlaid by some rules listed above it.

2. Rule  $r_2$  becomes redundant after  $r_3$  is removed. Note that  $r_2$  is the first matching rule for all packets where  $F_1 \in [51, 90]$ . However, if we remove  $r_2$  (assuming that  $r_3$  has been removed), the first matching rule for all those packets becomes  $r_4$  instead of  $r_2$ . This does not change the function of the firewall since both  $r_2$  and  $r_4$  have the same decision. We call  $r_2$  a downward redundant rule. A rule  $r$  in a firewall is *downward redundant* iff for each packet, whose first matching rule is  $r$ , the first matching rule below  $r$  has the same decision as  $r$ .

Redundant rules significantly degrade the performance of firewalls. A firewall maps a packet to the decision of the first rule that the packet matches using packet classification algorithms. A packet classification algorithm maps each packet to the right decision using an internal data structure built from a firewall of a sequence of rules. The fewer the rules in a firewall are, the faster a packet classification algorithm can map a packet to the right decision. To map a given packet to the decision of the first rule that the packet matches, according to the complexity bounds from computational geometry [15], the “best” software-based packet classification algorithm uses either  $O(n^d)$  space and  $O(\log n)$  time or  $O(n)$  space and  $O(\log^{d-1} n)$  time, where  $n$  is the total number of rules and  $d$  ( $d > 3$ ) is the total number of fields that the firewall examines for every packet. Clearly, for software-based packet classification algorithms, either space or running time grows quickly as the number of rules increases. Reducing the space that a software-based packet classification algorithm needs also helps to reduce the running time of the algorithm because small space consumption could enable the use of very limited on-chip cache to store the data structure of the algorithm. All in all, for software-based packet classification algorithms, it is advantageous to reduce the number of rules in a firewall. For hardware-based packet classification algorithms, it is also advantageous to reduce the number of rules in a firewall. Consider the example of a TCAM (Ternary Content Addressable Memory). A TCAM uses  $O(n)$  space and constant time in mapping a given packet to the decision of the first rule the packet matches. Moreover, TCAM consumes less power as the number of rules decreases.

### 1.3 Related Work

Previous work on firewalls has primarily focused on firewall design (see [5, 6, 10, 13, 8]) and firewall analysis (see [14, 16, 11, 12, 7]). None of these papers address the issue of redundant rules. The problem of detecting redundant rules only receives attention in [9, 2, 3, 4].

In [9], two special types of redundant rules are identified: backward redundant rules and forward redundant rules. A rule  $r$  in a firewall is backward redundant iff there exists another rule  $r'$  listed above  $r$  such that all packets that match  $r$

also match  $r'$ . Clearly, a backward redundant rule is an upward redundant rule, but not vice versa. For example, rule  $r_3$  in Figure 1 is upward redundant, but not backward redundant. A rule  $r$  in a firewall is forward redundant iff there exists another rule  $r'$  listed below  $r$  such that the following three conditions hold: (1) all packets that match  $r$  also match  $r'$ , (2)  $r$  and  $r'$  have the same decision, (3) for each rule  $r''$  listed between  $r$  and  $r'$ , either  $r$  and  $r''$  have the same decision, or no packet matches both  $r$  and  $r''$ . Clearly, a forward redundant rule is a downward redundant rule, but not vice versa. For example, rule  $r_2$  in Figure 1, assuming  $r_3$  has been removed previously, is downward redundant, but not forward redundant. It has been observed in [9] that 15% of the rules in real-life firewalls are backward redundant or forward redundant.

The redundant rules identified in [2,3,4] are similar to those identified in [9], except that for the case of backward redundant rules, they require that the two rules  $r$  and  $r'$  must have the same decision.

The bottom line is that the set of redundant rules identified by previous work is incomplete. In other words, given a firewall, after we remove the redundant rules identified in previous work, the firewall still possibly has redundant rules. So, how to detect all the redundant rules in a firewall? This is a hard problem and this problem has never been addressed previously.

#### 1.4 Our Contribution

In this paper, we solve the problem of detecting all redundant rules in a firewall. First, we give a necessary and sufficient condition for identifying all redundant rules. Based on this condition, we categorize redundant rules into upward redundant rules and downward redundant rules. Second, we present methods for detecting the two types of redundant rules respectively. Our methods make use of a tree representation of firewalls, which is called firewall decision trees.

Note that removing redundant rules can be done by firewall software internally. Therefore, the external firewall configuration, i.e., the original sequence of rules which is viewed by firewall administrators, would remain the same. In other words, the procedure of removing redundant rules can be transparent to firewall administrators. Also note that applying our procedure of removing redundant rules does not prevent a firewall administrator from updating a firewall configuration. When the configuration of a firewall is changed due to some rules being inserted, deleted or modified, firewall software always needs to rebuild its internal data structure from the new sequence of rules.

## 2 Firewall Redundant Rules

We define a *packet* over the fields  $F_1, \dots, F_d$  as a  $d$ -tuple  $(p_1, \dots, p_d)$  where each  $p_i$  is a value in the domain  $D(F_i)$  of field  $F_i$ , and each  $D(F_i)$  is an interval of nonnegative integers. For example, the domain of the source address in an IP packet is  $[0, 2^{32} - 1]$ . We use  $\Sigma$  to denote the set of all packets over fields  $F_1, \dots, F_d$ . It follows that  $\Sigma$  is a finite set and  $|\Sigma| = |D(F_1)| \times \dots \times |D(F_n)|$ .

A *firewall* over the fields  $F_1, \dots, F_d$  is a sequence of rules, and each rule is of the following format:

$$(F_1 \in S_1) \wedge \dots \wedge (F_d \in S_d) \rightarrow \langle decision \rangle$$

where each  $S_i$  is a nonempty subset of  $D(F_i)$  and  $\langle decision \rangle$  is either *accept* or *discard*. For simplicity, in the rest of this paper, we assume that all packets and all firewalls are over the fields  $F_1, \dots, F_d$ , if not otherwise specified.

Some existing firewall products, such as Linux's ipchains [1], require each  $S_i$  in a rule to be represented in a prefix format. An example of a prefix is 192.168.0.0/16, where 16 means that the prefix is the first 16 bits of 192.168.0.0. In this paper we use "set", instead of "prefix", to describe firewall rules for two reasons. First, sets and prefixes are algorithmically interconvertible. For example, the set  $\{2, 3, \dots, 8\}$  can be converted to 3 prefixes: 001\*, 01\*, 1000. Second, it is easier to argue the mathematical properties of sets than those of prefixes.

A packet  $(p_1, \dots, p_d)$  *matches* a rule  $(F_1 \in S_1) \wedge \dots \wedge (F_d \in S_d) \rightarrow \langle decision \rangle$  iff  $(p_1 \in S_1) \wedge \dots \wedge (p_d \in S_d)$  holds.

A sequence of rules  $\langle r_1, \dots, r_n \rangle$  is *comprehensive* iff for any packet  $p$  in  $\Sigma$ , there is at least one rule in  $\langle r_1, \dots, r_n \rangle$  that  $p$  matches. A sequence of rules needs to be comprehensive for it to serve as a firewall. From now on, we assume that each firewall is comprehensive. Henceforth, the predicate of the last rule in a firewall can always be replaced by  $(F_1 \in D(F_1)) \wedge \dots \wedge (F_d \in D(F_d))$  without changing the function of the firewall. In the rest of this paper, we assume that the predicate of the last rule in a firewall is  $(F_1 \in D(F_1)) \wedge \dots \wedge (F_d \in D(F_d))$ . It follows from this assumption that any postfix of a firewall is comprehensive, i.e., given a firewall  $\langle r_1, r_2, \dots, r_n \rangle$ , we know that  $\langle r_i, r_{i+1}, \dots, r_n \rangle$  is comprehensive for each  $i$ ,  $1 \leq i \leq n$ .

We use  $f(p)$  to denote the decision to which a firewall  $f$  maps a packet  $p$ . Two firewalls  $f$  and  $f'$  are equivalent, denoted  $f \equiv f'$ , iff for any packet  $p$  in  $\Sigma$ ,  $f(p) = f'(p)$  holds. This equivalence relation is symmetric, self-reflective, and transitive. Using the concept of equivalent firewalls, we define redundant rules as follows.

**Definition 1 (Redundant Rule).** A rule  $r$  is *redundant* in a firewall  $f$  iff the resulting firewall  $f'$  after removing rule  $r$  is equivalent to  $f$ .

Before introducing our redundancy theorem, we define two important concepts that are associated with each rule in a firewall: matching set and resolving set.

**Definition 2 (Matching Set and Resolving Set).** Consider a firewall  $f$  that consists of  $n$  rules  $\langle r_1, r_2, \dots, r_n \rangle$ . The *matching set* of a rule  $r_i$  in this firewall is the set of all packets that match  $r_i$ . The *resolving set* of a rule  $r_i$  in this firewall is the set of all packets that match  $r_i$ , but do not match any  $r_j$  where  $j < i$ .

For example, consider rule  $r_2$  in Figure 1: its matching set is the set of all the packets whose  $F_1$  field is in  $[40, 90]$ ; and its resolving set is the set of all the packets whose  $F_1$  field is in  $[51, 90]$ .

The matching set of a rule  $r_i$  is denoted  $M(r_i)$ , and the resolving set of a rule  $r_i$  is denoted  $R(r_i, f)$ . Note that the matching set of a rule depends only on the rule itself, while the resolving set of a rule depends both on the rule and on all the rules listed above it in a firewall.

The following theorem states several important properties of matching sets and resolving sets.

**Theorem 1 (Resolving Set Theorem).** Let  $f$  be any firewall that consists of  $n$  rules:  $\langle r_1, r_2, \dots, r_n \rangle$ . The following four conditions hold:

1. Equality:  $\bigcup_{j=1}^i M(r_j) = \bigcup_{j=1}^i R(r_j, f)$  for each  $i$ ,  $1 \leq i \leq n$
2. Dependency:  $R(r_i, f) = M(r_i) - \bigcup_{j=1}^{i-1} R(r_j, f)$  for each  $i$ ,  $1 \leq i \leq n$
3. Determinism:  $R(r_i, f) \cap R(r_j, f) = \emptyset$  for each  $i \neq j$
4. Comprehensiveness:  $\bigcup_{i=1}^n R(r_i, f) = \Sigma$  □

The redundancy theorem below gives a necessary and sufficient condition for identifying redundant rules. Note that we use the notation  $\langle r_{i+1}, r_{i+2}, \dots, r_n \rangle(p)$  to denote the decision to which the firewall  $\langle r_{i+1}, r_{i+2}, \dots, r_n \rangle$  maps packet  $p$ .

**Theorem 2 (Redundancy Theorem).** Let  $f$  be any firewall that consists of  $n$  rules:  $\langle r_1, r_2, \dots, r_n \rangle$ . A rule  $r_i$  is *redundant* in  $f$  iff one of the following two conditions holds:

1.  $R(r_i, f) = \emptyset$ ,
2.  $R(r_i, f) \neq \emptyset$ , and for any  $p$  that  $p \in R(r_i, f)$ ,  $\langle r_{i+1}, r_{i+2}, \dots, r_n \rangle(p)$  yields the same decision as that of  $r_i$ . □

Note that removing rule  $r_i$  from firewall  $f$  only possibly affects the decision of the packets in  $R(r_i, f)$ . If  $R(r_i, f) = \emptyset$ , then  $r_i$  is clearly redundant. If  $R(r_i, f) \neq \emptyset$ , and for any  $p$  that  $p \in R(r_i, f)$ ,  $\langle r_{i+1}, r_{i+2}, \dots, r_n \rangle(p)$  yields the same as that of  $r_i$ , then  $r_i$  is redundant because removing  $r_i$  does not affect the decision of the packets in  $R(r_i, f)$ .

The redundancy theorem allows us to categorize redundant rules into upward and downward redundant rules.

**Definition 3.** A rule that satisfies condition 1 in the redundancy theorem is called *upward redundant*. A rule that satisfies condition 2 in the redundancy theorem is called *downward redundant*.

Consider the example firewall  $f$  in Figure 1. Rule  $r_3$  is an upward redundant rule because  $R(r_3, f) = \emptyset$ . Let  $f'$  be the resulting firewall by removing rule  $r_3$  from  $f$ . Then rule  $r_2$  is downward redundant in  $f'$ .

### 3 Firewall Decision Trees and Rules

In [8], Firewall Decision Diagrams are proposed as a useful notation for specifying firewalls. In this paper, we use a special type of firewall decision diagrams, called Firewall Decision Trees (FDTs), as the core data structure for detecting redundant rules.

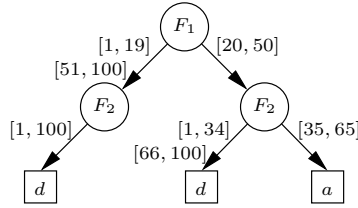
**Definition 4 (Firewall Decision Tree).** A Firewall Decision Tree  $t$  over fields  $F_1, \dots, F_d$  is a directed tree that has the following four properties:

1. Each node  $v$  in  $t$  has a label, denoted  $F(v)$ , such that

$$F(v) \in \begin{cases} \{F_1, \dots, F_d\} & \text{if } v \text{ is nonterminal,} \\ \{accept, discard\} & \text{if } v \text{ is terminal.} \end{cases}$$

2. Each edge  $e$  in  $t$  has a label, denoted  $I(e)$ , such that if  $e$  is an outgoing edge of node  $v$ , then  $I(e)$  is a nonempty subset of  $D(F(v))$ .
3. A directed path in  $t$  from the root to a terminal node is called a *decision path* of  $t$ . Each decision path contains  $d$  nonterminal nodes, and the  $i$ -th node is labelled  $F_i$  for each  $i$  that  $1 \leq i \leq d$ .
4. The set of all outgoing edges of a node  $v$  in  $t$ , denoted  $E(v)$ , satisfies the following two conditions:
  - (a) *Consistency:*  $I(e) \cap I(e') = \emptyset$  for any two distinct edges  $e$  and  $e'$  in  $E(v)$ ,
  - (b) *Completeness:*  $\bigcup_{e \in E(v)} I(e) = D(F(v))$  □

Figure 3 shows an example of an FDT over the two fields  $F_1$  and  $F_2$ , where  $D(F_1) = D(F_2) = [1, 100]$ . In the rest of this paper, including this example, we use “ $a$ ” as a shorthand for *accept* and “ $d$ ” as a shorthand for *discard*.



**Fig. 3.** An FDT

A decision path in an FDT  $t$  is represented by  $(v_1 e_1 \dots v_k e_k v_{k+1})$  where  $v_1$  is the root of  $t$ ,  $v_{k+1}$  is a terminal node of  $t$ , and each  $e_i$  is a directed edge from node  $v_i$  to node  $v_{i+1}$  in  $t$ . A decision path  $(v_1 e_1 \dots v_k e_k v_{k+1})$  in an FDT defines the following rule:

$$F_1 \in I(e_1) \wedge \dots \wedge F_n \in I(e_n) \rightarrow F(v_{k+1})$$

For example, the leftmost path in Figure 3 defines the following rule:

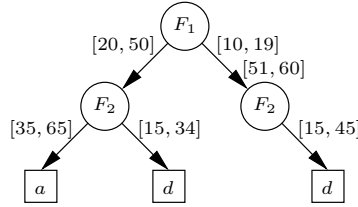
$$F_1 \in [1, 19] \cup [51, 100] \wedge F_2 \in [1, 100] \rightarrow d$$

We use  $\Gamma(t)$  to denote the set of all the rules defined by all the decision paths in FDT  $t$ . If we use  $t$  to denote the FDT in Figure 3, then  $\Gamma(t) = \{ (F_1 \in [1, 19] \cup [51, 100] \wedge F_2 \in [1, 100] \rightarrow d), (F_1 \in [20, 50] \wedge F_2 \in [66, 100] \rightarrow d), (F_1 \in [20, 50] \wedge F_2 \in [35, 65] \rightarrow a) \}$

$[1, 19] \cup [51, 100] \wedge (F_2 \in [1, 100]) \rightarrow d$ ,  $(F_1 \in [20, 50]) \wedge (F_2 \in [1, 34] \cup [66, 100]) \rightarrow d$ ,  $(F_1 \in [20, 50]) \wedge (F_2 \in [35, 65]) \rightarrow a$ .

For any packet  $p$ , there is one and only one rule in  $\Gamma(t)$  that  $p$  matches because of the consistency and completeness properties of FDT  $t$ . The semantics of an FDT  $t$  is that for any packet  $p$  in  $\Sigma$ ,  $t$  maps  $p$  to the decision of the only rule that  $p$  matches in  $\Gamma(t)$ . We use  $t(p)$  to denote the decision to which an FDT  $t$  maps a packet  $p$ . An FDT  $t$  and a sequence of rules  $f$  are equivalent, denoted  $t \equiv f$ , iff for any packet  $p$ ,  $t(p) = f(p)$  holds. Clearly, given an FDT  $t$ , any firewall that consists of all the rules in  $\Gamma(t)$  is equivalent to  $t$ . The order of the rules in such a firewall is immaterial because there are no overlapping rules in  $\Gamma(t)$ .

In the process of checking upward redundant rules, the data structure that we maintain is called a partial FDT. A *partial* FDT is a tree that may not have the completeness property of an FDT, but has all the other properties of an FDT. For example, Figure 4 shows a partial FDT.



**Fig. 4.** A partial FDT

We use  $\Gamma(t)$  to denote the set of all the rules defined by all the decision paths in a partial FDT  $t$ . For any packet  $p$  that  $p \in \bigcup_{r \in \Gamma(t)} M(r)$ , there is one and only one rule in  $\Gamma(t)$  that  $p$  matches. We use  $t(p)$  to denote the decision of the unique rule that  $p$  matches in  $\Gamma(t)$ .

Given a partial FDT  $t$  and a sequence of rules  $\langle r_1, r_2, \dots, r_k \rangle$  that may be not comprehensive, we say  $t$  is *equivalent* to  $\langle r_1, r_2, \dots, r_k \rangle$  iff the following two conditions hold:

1.  $\bigcup_{r \in \Gamma(t)} M(r) = \bigcup_{i=1}^k M(r_i)$ ,
2. for any packet  $p$  that  $p \in \bigcup_{r \in \Gamma(t)} M(r)$ ,  $t(p)$  is the same as the decision of the first rule that  $p$  matches in the sequence  $\langle r_1, r_2, \dots, r_k \rangle$ .

For example, the partial FDT in Figure 4 is equivalent to the sequence of rules  $\langle (F_1 \in [20, 50]) \wedge (F_2 \in [35, 65]) \rightarrow a, (F_1 \in [10, 60]) \wedge (F_2 \in [15, 45]) \rightarrow d \rangle$ .

## 4 Removing Upward Redundancy

In this section, we discuss how to remove upward redundant rules. By definition, a rule is upward redundant iff its resolving set is empty. Therefore, in order to

remove all upward redundant rules from a firewall, we need to calculate resolving set for each rule in the firewall. How to represent a resolving set? In this paper, we represent the resolving set of a rule by an effective rule set of the rule. An effective rule set of a rule  $r$  in a firewall  $f$  is a set of rules where the union of all the matching sets of these rules is exactly the resolving set of rule  $r$  in  $f$ . More precisely, an effective rule set of a rule  $r$  is defined as follows:

**Definition 5.** Let  $r$  be a rule in a firewall  $f$ . A set of rules  $\{r'_1, r'_2, \dots, r'_k\}$  is an *effective rule set* of  $r$  iff the following three conditions hold:

1.  $R(r, f) = \bigcup_{i=1}^k M(r'_i)$ ,
2.  $r'_i$  and  $r$  have the same decision for  $1 \leq i \leq k$ . □

For example, consider the firewall in Figure 1. Then,  $\{F_1 \in [1, 50] \rightarrow \text{accept}\}$  is an effective rule set of rule  $r_1$ ,  $\{F_1 \in [51, 90] \rightarrow \text{discard}\}$  is an effective rule set of rule  $r_2$ ,  $\emptyset$  is an effective rule set of rule  $r_3$ , and  $\{F_1 \in [91, 100] \rightarrow \text{discard}\}$  is an effective rule set of rule  $r_4$ . Clearly, once we obtain an effective rule set of a rule  $r$  in a firewall  $f$ , we know the resolving set of the rule  $r$  in  $f$ , and consequently know whether the rule  $r$  is upward redundant in  $f$ . Note that by the definition of an effective rule set, if one effective rule set of a rule  $r$  is empty, then any effective rule set of the rule  $r$  is empty. Based on the above discussion, we have the following upward redundancy theorem:

**Theorem 3 (Upward Redundancy Theorem).** A rule  $r$  is upward redundant in a firewall iff an effective rule set of  $r$  is empty. □

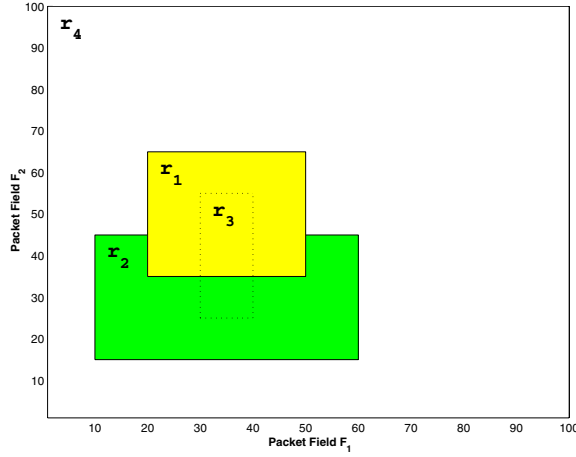
Based on the above upward redundancy theorem, the basic idea of our upward redundancy removal method is as follows: given a firewall  $\langle r_1, r_2, \dots, r_n \rangle$ , we calculate an effective rule set for each rule from  $r_1$  to  $r_n$ . If the effective rule set calculated for a rule  $r_i$  is empty, then  $r_i$  is upward redundant and is removed. Now the problem is how to calculate an effective rule set for every rule in a firewall.

An effective rule set for each rule in a firewall is calculated with the help of partial FDTs. Consider a firewall that consists of  $n$  rules  $\langle r_1, r_2, \dots, r_n \rangle$ . We first build a partial FDT, denoted  $t_1$ , that is equivalent to the sequence  $\langle r_1 \rangle$ , and calculates an effective rule set, denoted  $E_1$ , of rule  $r_1$ . Then we transform the partial FDT  $t_1$  to another partial FDT, denoted  $t_2$ , that is equivalent to the sequence  $\langle r_1, r_2 \rangle$ , and during the transformation process, we calculate an effective rule set, denoted  $E_2$ , of rule  $r_2$ . The same transformation process continues until we reach  $r_n$ . When we finish, an effective rule set is calculated for every rule.

Here we use  $t_i$  to denote the partial FDT that we constructed from the rule sequence  $\langle r_1, r_2, \dots, r_i \rangle$ , and  $E_i$  to denote the effective rule set that we calculated for rule  $r_i$ . By the following example, we show the process of transforming the partial FDT  $t_i$  to the partial FDT  $t_{i+1}$ , and the calculation of  $E_{i+1}$ . Consider the firewall in Figure 5 over fields  $F_1$  and  $F_2$ , where  $D(F_1) = D(F_2) = [1, 100]$ . Figure 6 shows the geometric representation of this firewall, where each rule is represented by a rectangle. From Figure 6, we can see that rule  $r_3$  is upward

$$\begin{aligned}
r_1 &: (F_1 \in [20, 50]) \wedge (F_2 \in [35, 65]) \rightarrow a \\
r_2 &: (F_1 \in [10, 60]) \wedge (F_2 \in [15, 45]) \rightarrow d \\
r_3 &: (F_1 \in [30, 40]) \wedge (F_2 \in [25, 55]) \rightarrow a \\
r_4 &: (F_1 \in [1, 100]) \wedge (F_2 \in [1, 100]) \rightarrow d
\end{aligned}$$

**Fig. 5.** A firewall of 4 rules

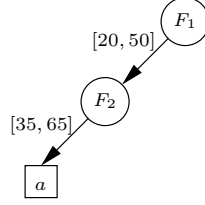


**Fig. 6.** Geometric representation of the rules in Figure 5

redundant because  $r_3$ , whose area is marked by dashed lines, is totally overlaid by rules  $r_1$  and  $r_2$ . Later we will see that the effective rule set calculated by our upward redundancy removal method for rule  $r_3$  is indeed an empty set.

Figure 7 shows a partial FDT  $t_1$  that is equivalent to  $\langle r_1 \rangle$  and the effective rule set  $E_1$  calculated for rule  $r_1$ . In this figure, we use  $v_1$  to denote the node with label  $F_1$ ,  $e_1$  to denote the edge with label  $[20, 50]$ , and  $v_2$  to denote the node with label  $F_2$ .

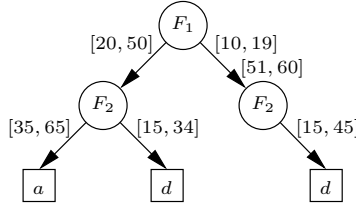
Now we show how to append rule  $r_2$  to  $t_1$  in order to get a partial FDT  $t_2$  that is equivalent to  $\langle r_1, r_2 \rangle$ , and how to calculate an effective rule set  $E_2$  for rule  $r_2$ . Rule  $r_2$  is  $(F_1 \in [10, 60]) \wedge (F_2 \in [15, 45]) \rightarrow d$ . We first compare the set  $[10, 60]$  with the set  $[20, 50]$  labelled on the outgoing edge of  $v_1$ . Since  $[10, 60] - [20, 50] = [10, 19] \cup [51, 60]$ ,  $r_2$  is the first matching rule for all the packets that satisfy  $F_1 \in [10, 19] \cup [51, 60] \wedge F_2 \in [15, 45]$ , so we add one outgoing edge  $e$  to  $v_1$ , where  $e$  is labeled  $[10, 19] \cup [51, 60]$  and  $e$  points to the path built from  $F_2 \in [15, 45] \rightarrow d$ . The rule defined by the decision path containing  $e$ , i.e.,  $F_1 \in [10, 19] \cup [51, 60] \wedge F_2 \in [15, 45] \rightarrow d$ , should be put in  $E_2$  because for all packets that match this rule,  $r_2$  is their first matching rule. Since  $[20, 50] \subset [10, 60]$ ,  $r_2$  is possibly the first matching rule for a packet that satisfies  $F_1 \in [20, 50]$ . So we further compare the set  $[35, 65]$  labeled on the outgoing edge of  $v_2$  with the set



$$E_1 = \{F_1 \in [20, 50] \wedge F_2 \in [35, 65] \rightarrow a\}$$

**Fig. 7.** Partial FDT  $t_1$  and the effective rule set  $E_1$  calculated for rule  $r_1$  in Figure 5

$[15, 45]$ . Since  $[15, 45] - [35, 65] = [15, 34]$ , we add a new edge  $e'$  to  $v_2$ , where  $e'$  is labeled  $[15, 34]$  and  $e'$  points to a terminal node labeled  $d$ . Similarly, we add the rule,  $F_1 \in [20, 50] \wedge F_2 \in [15, 34] \rightarrow d$ , defined by the decision path containing the new edge  $e'$  into  $E_2$ . The partial FDT  $t_2$  and the effective rule set  $E_2$  of rule  $r_2$  is shown in Figure 8.



$$E_2 = \{F_1 \in [10, 19] \cup [51, 60] \wedge F_2 \in [15, 45] \rightarrow d, \\ F_1 \in [20, 50] \wedge F_2 \in [15, 34] \rightarrow d\}$$

**Fig. 8.** Partial FDT  $t_2$  and the effective rule set  $E_2$  calculated for rule  $r_2$  in Figure 5

Let  $f$  be any firewall that consists of  $n$  rules:  $\langle r_1, r_2, \dots, r_n \rangle$ . The partial FDT that is equivalent to  $\langle r_1 \rangle$  consists of only one decision path that defines the rule  $r_1$ .

Suppose that we have constructed a partial FDT  $t_i$  that is equivalent to the sequence  $\langle r_1, r_2, \dots, r_i \rangle$ , and have calculated an effective rule set for each of these  $i$  rules. Let  $v$  be the root of  $t_i$ , and assume  $v$  has  $k$  outgoing edges  $e_1, e_2, \dots, e_k$ . Let rule  $r_{i+1}$  be  $(F_1 \in S_1) \wedge (F_2 \in S_2) \wedge \dots \wedge (F_d \in S_d) \rightarrow \langle decision \rangle$ . Next we consider how to transform the partial FDT  $t_i$  to a partial FDT, denoted  $t_{i+1}$ , that is equivalent to the sequence  $\langle r_1, r_2, \dots, r_i, r_{i+1} \rangle$ , and during the transformation process, how to calculate an effective rule set, denoted  $E_{i+1}$ , for rule  $r_{i+1}$ .

First, we examine whether we need to add another outgoing edge to  $v$ . If  $S_1 - (I(e_1) \cup I(e_2) \cup \dots \cup I(e_k)) \neq \emptyset$ , we need to add a new outgoing edge  $e_{k+1}$  with label  $S_1 - (I(e_1) \cup I(e_2) \cup \dots \cup I(e_k))$  to  $v$ . This is because any packet, whose  $F_1$  field satisfies  $S_1 - (I(e_1) \cup I(e_2) \cup \dots \cup I(e_k))$ , does not match any of the first  $i$  rules, but matches  $r_{i+1}$  provided that the packet also satisfies

$(F_2 \in S_2) \wedge (F_3 \in S_3) \wedge \dots \wedge (F_d \in S_d)$ . The new edge  $e_{k+1}$  points to the root of the path that is built from  $(F_2 \in S_2) \wedge (F_3 \in S_3) \wedge \dots \wedge (F_d \in S_d) \rightarrow \langle decision \rangle$ . The rule  $r$ ,  $(F_1 \in S_1 - (I(e_1) \cup I(e_2) \cup \dots \cup I(e_k))) \wedge (F_2 \in S_2) \wedge \dots \wedge (F_d \in S_d) \rightarrow \langle decision \rangle$ , defined by the decision path containing the new edge  $e_{k+1}$  has the property  $M(r) \subseteq R(r_{i+1}, f)$ . Therefore, we add rule  $r$  to  $E_i$ .

Second, we compare  $S_1$  and  $I(e_j)$  for each  $j$  ( $1 \leq j \leq k$ ) in the following three cases:

1.  $S_1 \cap I(e_j) = \emptyset$ : In this case, we skip edge  $e_j$  because any packet whose value of field  $F_1$  is in set  $I(e_j)$  doesn't match  $r_{i+1}$ .
2.  $S_1 \cap I(e_j) = I(e_j)$ : In this case, for a packet  $p$  whose value of field  $F_1$  is in set  $I(e_j)$ , the first rule that  $p$  matches may be one of the first  $i$  rules, and may be rule  $r_{i+1}$ . So we append  $(F_2 \in S_2) \wedge (F_3 \in S_3) \wedge \dots \wedge (F_d \in S_d) \rightarrow \langle decision \rangle$  to the subtree rooted at the node that  $e_j$  points to in a similar fashion.
3.  $S_1 \cap I(e_j) \neq \emptyset$  and  $S_1 \cap I(e_j) \neq I(e_j)$ : In this case, we split edge  $e$  into two edges:  $e'$  with label  $I(e_j) - S_1$  and  $e''$  with label  $I(e_j) \cap S_1$ . Then we make two copies of the subtree rooted at the node that  $e_j$  points to, and let  $e'$  and  $e''$  point to one copy each. Thus we can deal with  $e'$  by the first case, and  $e''$  by the second case.

In the process of appending rule  $r_{i+1}$  to partial FDT  $t_i$ , each time that we add a new edge to a node in  $t_i$ , the rule defined by the decision path containing the new edge is added to  $E_{i+1}$ . After the partial FDT  $t_i$  is transformed to  $t_{i+1}$ , according to the transformation process, the rules in  $E_{i+1}$  satisfy the following two conditions: (1) the union of all the matching sets of these rules is the resolving set of  $r_{i+1}$ , (2) all these rules have the same decision as  $r_{i+1}$ . Therefore,  $E_{i+1}$  is an effective rule set of rule  $r_{i+1}$ .

By applying our upward redundancy removal method to the firewall in Figure 5, we get an effective rule set for each rule as shown in Figure 9. Note that  $E_3 = \emptyset$ , which means that rule  $r_3$  is upward redundant, therefore  $r_3$  is removed.

$$\begin{aligned}
1 : E_1 &= \{F_1 \in [20, 50] \wedge F_2 \in [35, 65] && \rightarrow a\}; \\
2 : E_2 &= \{F_1 \in [10, 19] \cup [51, 60] \wedge F_2 \in [15, 45] && \rightarrow d \\
&\quad F_1 \in [20, 50] \wedge F_2 \in [15, 34] && \rightarrow d\}; \\
3 : E_3 &= \emptyset; \\
4 : E_4 &= \{ \\
&\quad F_1 \in [1, 9] \cup [61, 100] \wedge F_2 \in [1, 100] && \rightarrow d \\
&\quad F_1 \in [20, 29] \cup [41, 50] \wedge F_2 \in [1, 14] \cup [66, 100] && \rightarrow d \\
&\quad F_1 \in [30, 40] \wedge F_2 \in [1, 14] \cup [66, 100] && \rightarrow d \\
&\quad F_1 \in [10, 19] \cup [51, 60] \wedge F_2 \in [1, 14] \cup [46, 100] && \rightarrow d\}
\end{aligned}$$

**Fig. 9.** Effective rule sets calculated for the firewall in Figure 5

## 5 Removing Downward Redundancy

One particular advantage of detecting and removing upward redundant rules before detecting and removing downward redundant rules in a firewall is that an effective rule set for each rule is calculated by the upward redundancy removal method; therefore, we can use the effective rule set of a rule to check whether the rule is downward redundant. Note that knowing an effective rule set of a rule equals knowing the resolving set of the rule.

Our method for removing downward redundant rules is based on the following theorem.

**Theorem 4.** Let  $f$  be any firewall that consists of  $n$  rules:  $\langle r_1, r_2, \dots, r_n \rangle$ . Let  $t'_i$  ( $2 \leq i \leq n$ ) be an FDT that is equivalent to the sequence of rules  $\langle r_i, r_{i+1}, \dots, r_n \rangle$ . The rule  $r_{i-1}$  with the effective rule set  $E_{i-1}$  is downward redundant in  $f$  iff for each rule  $r$  in  $E_{i-1}$  and for each decision path  $(v_1e_1v_2e_2 \dots v_de_dv_{d+1})$  in  $t'_i$  where rule  $r$  overlaps the rule that is defined by this decision path, the decision of  $r$  is the same as the label of the terminal node  $v_{d+1}$ .

Now we consider how to construct an FDT  $t'_i$ ,  $2 \leq i \leq n$ , that is equivalent to the sequence of rules  $\langle r_i, r_{i+1}, \dots, r_n \rangle$ . The FDT  $t'_n$  can be built from rule  $r_n$  in the same way that we build a path from a rule in the upward redundancy removal method.

Suppose we have constructed an FDT  $t'_i$  that is equivalent to the sequence of rules  $\langle r_i, r_{i+1}, \dots, r_n \rangle$ . First, we check whether rule  $r_{i-1}$  is downward redundant by Theorem 4. If rule  $r_{i-1}$  is downward redundant, then we remove  $r_i$ , rename the FDT  $t'_i$  to be  $t'_{i-1}$ , and continue to check whether  $r_{i-2}$  is downward redundant. If rule  $r_{i-1}$  is not downward redundant, then we append rule  $r_{i-1}$  to the FDT  $t'_i$  such that the resulting tree is an FDT, denoted  $t'_{i-1}$ , that is equivalent to the sequence of rules  $\langle r_{i-1}, r_i, \dots, r_n \rangle$ . This procedure of transforming an FDT by appending a rule is similar to the procedure of transforming a partial FDT in the upward redundancy removal method. The above process continues until we reach  $r_1$ ; therefore, all downward redundant rules are detected and removed.

Applying our downward redundancy removal method to the firewall in Figure 5, assuming  $r_3$  has been removed, rule  $r_2$  is detected to be downward redundant, therefore  $r_2$  is removed. The FDT in Figure 3 is the resulting FDT by appending rule  $r_1$  to the FDT that is equivalent to  $\langle r_4 \rangle$ .

## 6 Concluding Remarks

We make two major contributions in this paper. First, we give a necessary and sufficient condition for identifying all redundant rules, based on which we categorize redundant rules into upward redundant rules and downward redundant rules. Second, we present methods for detecting the two types of redundant rules respectively. Our methods make use of a tree representation of firewalls, which is called firewall decision trees.

The results in this paper can be extended for use in many systems where a system can be represented by a sequence of rules. Examples of such systems are rule-based systems in the area of artificial intelligence and access control in the area of databases. In these systems, we can extend the results in this paper to remove redundant rules and thereby make the systems more efficient.

## References

1. ipchains, <http://www.tldp.org/howto/ipchains-howto.html>.
2. E. Al-Shaer and H. Hamed. Firewall policy advisor for anomaly detection and rule editing. In *IEEE/IFIP Integrated Management IM'2003*, pages 17–30, March 2003.
3. E. Al-Shaer and H. Hamed. Management and translation of filtering security policies. In *IEEE International Conference on Communications*, pages 256–260, May 2003.
4. E. Al-Shaer and H. Hamed. Discovery of policy anomalies in distributed firewalls. In *IEEE INFOCOM'04*, pages 2605–2616, March 2004.
5. Y. Bartal, A. J. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. In *Proceeding of the IEEE Symposium on Security and Privacy*, pages 17–31, 1999.
6. Y. Bartal, A. J. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. *Technical Report EES2003-1, Dept. of Electrical Engineering Systems, Tel Aviv University*, 2003.
7. M. Frantzen, F. Kerschbaum, E. Schultz, and S. Fahmy. A framework for understanding vulnerabilities in firewalls using a dataflow model of firewall internals. *Computers and Security*, 20(3):263–270, 2001.
8. M. G. Gouda and A. X. Liu. Firewall design: consistency, completeness and compactness. In *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS'04)*, pages 320–327.
9. P. Gupta. *Algorithms for Routing Lookups and Packet Classification*. PhD thesis, Stanford University, 2000.
10. J. D. Guttman. Filtering postures: Local enforcement for global policies. In *Proceedings of IEEE Symp. on Security and Privacy*, pages 120–129, 1997.
11. S. Hazelhurst, A. Attar, and R. Sinnappan. Algorithms for improving the dependability of firewall and filter rule lists. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'00)*, pages 576–585, 2000.
12. S. Kamara, S. Fahmy, E. Schultz, F. Kerschbaum, and M. Frantzen. Analysis of vulnerabilities in internet firewalls. *Computers and Security*, 22(3):214–232, 2003.
13. A. X. Liu and M. G. Gouda. Diverse firewall design. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'04)*, pages 595–604, June 2004.
14. A. Mayer, A. Wool, and E. Ziskind. Fang: A firewall analysis engine. In *Proceedings of IEEE Symp. on Security and Privacy*, pages 177–187, 2000.
15. M. H. Overmars and A. F. van der Stappen. Range searching and point location among fat objects. *Journal of Algorithms*, 21(3):629–656.
16. A. Wool. Architecting the lumeta firewall analyzer. In *Proceedings of the 10th USENIX Security Symposium*, pages 85–97, August 2001.