SPECIFICATION BASED FIREWALL TESTING

THESIS

Present to the Graduate Council
of Texas State University-San Marcos
in Partial Fulfillment
of the Requirements

for the Degree of

Master of SCIENCE

by

Huibo Heidi Ma, B.S.

San Marcos, Texas
May 2004

Dedicated with love and respect

to my parents,

Rongguan Ma and Yingmin Yu,

to whom I owe

all that I am and all that I have accomplished.

Acknowledgments

This thesis represents the culmination of my career as master student and at this juncture, I would like to pause and thank the many people whose support and guidance brought me here.

First and foremost, I would like to thank my advisor, Dr. Anne Hee Hiong Ngu for her continued guidance and support. I truly appreciate the patience and respect that she has given me.

I am extremely grateful to the members in my thesis committee, Dr. Xiao Chen and Prof. Don Shafer, for being a constant source of encouragement and advice.

The development of the thesis benefit greatly from the discussions with Dr. M. G. Gouda and Alex X. Liu. Most of ideas in this thesis came out our technical discussions.

This manuscript was submitted on April 27,2004.

HUIBO HEIDI MA

Texas State University-San Marcos

May 2004

Contents

List of Figures

SPECIFICATION BASED FIREWALL TESTING

HUIBO HEIDI MA, M.A.

Texas State University-San Marcos, 2004

Supervisor Professor: Anne Hee Hiong Ngu

Firewalls are crucial elements in network security, and have been widely deployed in most businesses and institutions for securing private networks. A firewall consists of a sequence of rules. The function of a firewall is to examine each incoming and outgoing packet and decide to either accept the packet (i.e., allow it to proceed) or discard the packet based on the sequence of rules. The decision made by a firewall for a packet is the decision of the first rule that the packet matches.

As a safety-critical system, a firewall needs to be correctly implemented by a sequence of rules according to its specification. However, since the number of rules in a firewall may be large and the rules may conflict, a firewall often contains errors that make the firewall inconsistent with its specification.

To check whether the firewall implementation of a sequence of rules is consistent with its specification or not, a firewall designer usually need to figure out the answers to the queries such as "which computers in the private network can receive BOOTP packets from the outside Internet?". We call the process of testing a firewall by issuing such test queries *specification based firewall testing.*

The technical challenge in specification based firewall testing is how to answer the test queries based on a firewall specification. To solve this problem, in this thesis, we propose a firewall testing algorithm based on a data structure called

Firewall Decision Diagram proposed in [11]. Given a firewall of a sequence of rules, we at first construct an equivalent firewall decision diagram from the sequence of rules by the construction algorithm in Chapter 3. Then given each firewall testing query, the firewall decision diagram is used as the core data structure for answering the query by the firewall testing algorithm in Chapter 4.

The experimental results show that our firewall testing algorithm is very efficient. Even given a firewall of 5000 rules, it takes less than 4 seconds for the firewall testing algorithm to answer a firewall testing query.

Chapter 1

INTRODUCTION

Firewalls are the corner stones of network security. A firewall is a security guard 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. For each incoming and outgoing packet, a firewall maps the packet to a decision by examining the values of these fields and the physical network interface that the packet arrives on. The decision can be *accept* or *discard*, or a combination of these with other options such as the logging option. For the sake of brevity, we assume that each packet has a field containing the information of the network interface on which a packet arrives, and the decision for a packet is either *accept* or *discard*.

A firewall consists of a sequence of rules. Each rule is of the form

$$\langle \textit{predicate} \rangle \rightarrow \langle \textit{decision} \rangle$$

, where the $\langle \textit{predicate} \rangle$ is a boolean expression over the packet fields, and the $\langle \textit{decision} \rangle$ is either *accept* or *discard*. Figure 1.1 shows an example of a firewall on the gateway router in Figure 1.2. This gateway router has two interfaces: interface 0, which connects the gateway router to the outside Internet, and interface 1, which connects the gateway router to the inside local network. In this thesis, we use the following shorthand: I (Interface), S (Source IP), D (Destination IP), N (Destination Port), P (Protocol Type), a (Accept), d (Discard), and we assume that the value of the protocol type field of a packet is either 0 (TCP) or 1 (UDP). Also, in this thesis, we use "all" to denote the domain of the corresponding packet field. For example, $(S \in \{all\})$ means the packet field S can take any value from the domain of S. We use α to denote the integer formed by the four bytes of the IP address of the mail server, i.e., 192.168.0.1, and similarly β for the IP address 192.168.0.2 used by a local host inside the private network, and γ for the IP address of a previously known malicious host outside of the private network.

1. $r_1 :\ (I \in \{0\}) \wedge (S \in \{all\}) \wedge (D \in \{\alpha\}) \wedge (N \in \{25\}) \wedge (P \in \{0\}) \rightarrow a$
   This rule allows incoming SMTP packets to proceed to the mail server.
2. $r_2 :\ (I \in \{0\}) \wedge (S \in \{\gamma\}) \wedge (D \in \{all\}) \wedge (N \in \{all\}) \wedge (P \in \{all\}) \rightarrow d$
   This rule discards incoming packets from a previously known malicious host.
3. $r_3 :\ (I \in \{0\}) \wedge (S \in \{all\}) \wedge (D \in \{\beta\}) \wedge (N \in \{all\}) \wedge (P \in \{all\}) \rightarrow a$
   This rule allows host β unrestricted communication with outside Internet.
4. $r_4 :\ (I \in \{0\}) \wedge (S \in \{all\}) \wedge (D \in \{all\}) \wedge (N \in \{67, 68\}) \wedge (P \in \{1\}) \rightarrow d$
   This rule discards incoming BOOTP packets. The BOOTP protocol is used by local computers to obtain IP addresses Therefore, the service should be banned for outside computers.
5. $r_5 :\ (I \in \{all\}) \wedge (S \in \{all\}) \wedge (D \in \{all\}) \wedge (N \in \{all\}) \wedge (P \in \{all\}) \rightarrow a$
   This last rule allows any incoming and outgoing packets to proceed.

Figure 1.1: A simple firewall

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. The rules in a firewall may overlap and conflict. Two rules *overlap* iff there is at least one packet that matches both rules. Two rules *conflict* iff these two rules overlap and also have different decisions. For example, the two rules r_1 and r_2 in Figure 1.1 conflict. For each incoming and outgoing packet, a firewall maps it to the decision of the first (i.e., highest priority) rule that the packet matches.

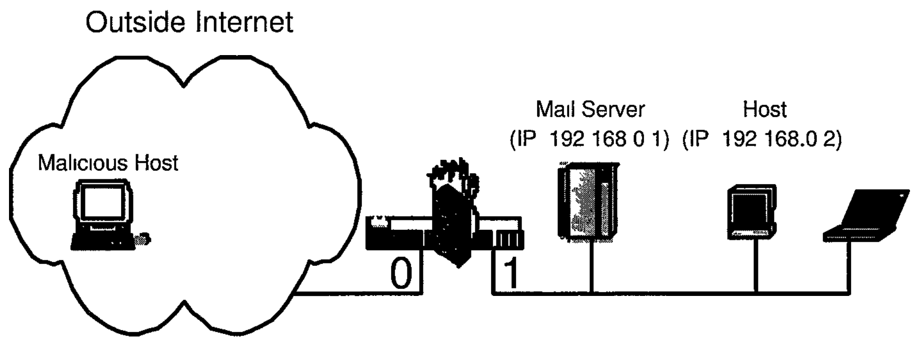

Figure 1.2: A simple network

Due to the conflicts among rules and the resulting order-sensitivity of the rules, the correctness of a firewall becomes hard to guarantee and the function of a firewall becomes difficult to analyze. The meaning of a rule cannot be understood without examining all the rules listed before it. For example, the meaning of the rule r_2 in Figure 1.1 is not exactly discarding all incoming packets from a previously discovered malicious host γ because γ can send SMTP packet to the mail server by the rule r_1.

Although the design of firewalls has gained some attention in the research community of network security (see [6, 11, 12]), the testing of firewalls has not been thoroughly studied. The previous work on firewall testing, such as [16, 15], are all based on injecting real packets to test a firewall. Such firewall testing methods are very inefficient. For example, assuming the specification of the firewall in Figure 1.2 requires that all the computers in the outside Internet, except the malicious host discovered previously, can send emails to the mail server in the private network, testing whether a firewall satisfies this requirement or not by sending bogus packets pretending originated from all the hosts outside the private network is very inefficient. In this thesis, we assume a firewall functions correctly as long as the sequence of rules of the firewall is consistent with the specification of the firewall.

To test the function a firewall, an effective way is to test the firewall based on its specification. For example, if the specification of a firewall requires that no computers in the private network should be reached by the BOOTP packets from the outside Internet, we can issue a query 'which computers in the private network can receive BOOTP packets from the outside Internet?". If the answer to this query is an empty set, then we can conclude that the firewall satisfies this requirement of its specification. If the answer to this query is a nonempty set, then we know that the firewall fails to satisfy this requirement, and we can further investigate why the computers in the nonempty set can receive BOOTP packets. We call these queries *firewall testing queries*, and the process of testing a firewall by such queries *specification based firewall testing*

Specification based firewall testing is useful in a variety of ways. For example:

1. Verifying the correctness of a firewall. Whether a firewall satisfies its specification can be checked by issuing firewall testing queries.

2. Maintaining a firewall. For a firewall administrator, issuing firewall testing queries is part of the daily maintenance activities. For example, if a computer in a private network is found being attacked, the firewall administrator can issue queries to check who else are also vulnerable to the same type of attack.

3. Debugging a firewall. Although no debugging tools have been seen for designing firewalls, we believe that specification based firewall testing should be a valuable component for a firewall debugger. In the process of designing a firewall, the designer can use firewall testing queries to detect the design errors in the firewall by checking whether the answers are consistent with the firewall specification.

4. Understanding a legacy firewall. The firewall for an enterprise private network typically consists of a large number (say hundreds) of rules which are written by different administrator in different times. The first thing for a

new administrator who just takes over a legacy firewall is to understand the firewall, which can be assisted by issuing firewall testing queries.

Although analyzing firewalls by queries was previously discussed in [4, 8, 14], the problem of how to efficiently process a firewall query remains unsolved. We need an algorithm that can answer a firewall testing query in at most a few seconds. The processing of a firewall testing query must be fast in order to interact with a human user. Our solution to this problem is an efficient firewall testing algorithm. The core data structure of our firewall testing algorithm is a firewall decision diagram. Given a firewall of a sequence of rules, we first construct an equivalent firewall decision diagram from the sequence of rules by the construction algorithm in Chapter 3. Then the firewall decision diagram is used as the core data structure for efficiently processing firewall testing queries.

The experimental results show that our firewall testing algorithm is very efficient. Even given a firewall of 5000 rules, it takes less than 4 seconds for the firewall testing algorithm to answer a firewall testing query.

The rest of this thesis is organized as follows. We start with a detailed examination of previous related work, and compare it with our approach in Chapter 2. Then in Chapter 3, we introduce the algorithm for converting a firewall of a sequence of rules to an equivalent firewall decision diagram The firewall testing algorithm is presented in Chapter 4. In Chapter 5, we show the experimental results about the efficiency of the construction algorithm and the firewall testing algorithm. The concluding remarks are given in Chapter 6.

Chapter 2

RELATED WORK

Analyzing firewalls by queries was previously discussed in [4, 8, 14]. In [4], a query language for requesting information from a firewall is proposed. However, the query language proposed in [4] is limited in terms of its expressive power. For example, it cannot describe the queries such as 'which computers of the outside Internet cannot send SMTP packets to the mail server in the private network?". In addition, how a query is processed based on a sequence of rules is not presented in [4]. Some ad-hoc firewall testing queries are discussed in [8, 14], however, no algorithm on processing these queries is presented. Therefore, how fast and scalable that the firewall testing queries can be processed in [8, 14] is not clear. As we mention earlier, the efficiency of firewall testing query processing is important because a firewall administrator needs to interact with the firewall testing query engine.

There are some tools currently available for network vulnerability testing, such as Satan [9, 10] and Nessus [19]. These vulnerability testing tools scan a private network based on the current publicly known attacks, rather than the requirement specification of a firewall. Although these tools can possibly catch errors that allow illegitimate access to the private network, it cannot find the errors that disable legitimate communication between the private network and the outside Internet.

Instead of firewall testing, ensuring the correctness of a firewall by conflicts detection was discussed in [18, 13, 7, 5]. Similar to conflicts detection, six types of so-called "anomalies" are defined in [1, 2, 3] Examining each conflict or anomaly is helpful in reducing errors. However, conflict detection creates more problems than it solves. At first, the number of conflicts in a sequence of rules can be huge. For a firewall with n rules, the number of conflicts is $O(n^2)$. For each conflict, [13] proposes that the firewall administrator check whether the two rules need to be swapped or a new rule needs to be added to resolve the conflict. This manual checking is highly unreliable because each of the two conflicting rules have to be understood in the current order of the firewall, which may be not correct. Since the number of conflicts in a firewall can be huge, this manual checking for each conflict can be tremendous work for the firewall administrator. Second, if a new rule is added for each conflict, as proposed in [13], the firewall can grow extremely big because each new rule added to the firewall may conflict with existing rules and therefore create $O(n)$ new conflicts. When the number of rules in a firewall becomes big, the packet filtering performance will degrade dramatically; also, the firewall becomes too hard to understand. Third, conflict detection does not solve the incompleteness and redundancy problems.

The previous work on firewall testing, such as [16, 15], are all based on injecting real packets to test a firewall. Testing firewalls by injecting packets are either inefficient or incomplete. As an example, consider the firewall in Figure 1.2, assuming the specification of the firewall requires that all the computers in the outside Internet, except the malicious host discovered previously, can send emails to the mail server in the private network. There are two strategies to inject packets: (1) injecting bogus packets pretending originated from *all* the hosts outside the private network; (2) injecting bogus packets pretending originated from *some* the hosts outside the private network. Clearly, the first strategy of injecting packets is extremely inefficient, and the second strategy of injecting packets cannot cover all possible host IP addresses, which make the testıng incomplete.

Chapter 3

FDD

Firewall Decision Diagrams are proposed in [11] for specifying firewalls. In this Chapter, we discuss how to construct an equivalent Firewall Decision Diagram from a firewall of a sequence of rules. In Chapter 4, we will see that Firewall Decision Diagrams are used as the core data structures for processing firewall testing queries.

A *Firewall Decision Diagram* (FDD) f over fields $F_1, \cdots, F_d$ is an acyclic and directed graph that has the following five properties:

1. There is exactly one node in f that has no incoming edges and it is called the *root* of f. The nodes in f that have no outgoing edges are called *terminal* nodes of f.

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

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

3. Each edge e in f has a label, denoted $I(e)$, such that if e is an outgoing edge of node v, then

$$I(e) \subseteq D(F(v)).$$

4. A directed path in f from the root to a terminal node is called a *decision path* of f. No two nodes on a decision path have the same label.

5. The set of all outgoing edges of a node v in f, 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 3.1 shows an FDD over two fields F_1 and F_2, where $D(F_1) = D(F_2) = [0, 9]$. The label of each edge is represented by one or more non-overlapping intervals whose union is the label of the edge. For example, the label of the rightmost outgoing edge of the root is $\{0, 1, 2, 3, 8, 9\}$, which is represented by two the intervals $[0, 3]$ and $[7, 9]$.

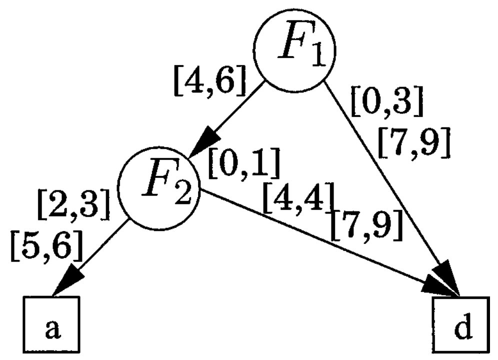

Figure 3.1: An FDD example

A firewall over fields $F_1, \cdots, F_d$ is a firewall in which all the packets pass through it are tuples of these n fields. For the rest of this paper, we assume that all firewalls are over these n fields and all packets are tuples of these n fields, if not otherwise specified.

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

$$F_1 \in S_1 \wedge \cdots \wedge F_n \in S_n \;\rightarrow\; F(v_{k+1})$$

where

$$S_i = \begin{cases} I(e_j) & \text{if there is a node } v_j \text{ in the decision} \\ & \text{path that is labelled with field } F_i, \\ & \\ D(F_i) & \text{if no node in the decision path is} \\ & \text{labelled with field } F_i. \end{cases}$$

For an FDD f, we use S_f to represent the set of all the rules defined by all the decision paths of f. For any packet p, there is one and only one rule in S_f that p matches because of the consistency and completeness properties; therefore, f maps p to the decision of the only rule that p matches.

As an example, let f denote the FDD in Figure 3.1. Therefore, S_f consists of the following three rules in Figure 3.2. Note that these rules are non-overlapping.

1. $r_1 : \ F_1 \in [4,6] \wedge F_2 \in [2,3] \cup [5,6] \rightarrow \ a$
2. $r_2 : \ F_1 \in [4,6] \wedge F_2 \in [0,1] \cup [4,4] \cup [7,9] \rightarrow \ d$
3. $r_3 : \ F_1 \in [0,3] \cup [7,9] \wedge F_2 \in [0,9] \rightarrow \ d$

Figure 3.2: Rules from an FDD

Given an FDD f, any sequence of rules that consists of all the rules in S_f is equivalent to f. The order of the rules in such a firewall is immaterial because the rules in S_f are non-overlapping. For example, the sequence of rules $\langle r_1, r_2, r_3 \rangle$ that consists of all the rules in Figure 3.2 is equivalent to the FDD in Figure 3.1.

Given a sequence of rules, how to construct an equivalent FDD? Next we discuss how to construct an equivalent FDD from a sequence of rules $\langle r_1, \ \cdots, \ r_n \rangle$, where each rule is of the format $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \rightarrow$ *decision*. Note that all the d packet fields appear in the predicate of each rule, and they appear in the same order. A similar algorithm for constructing FDDs has been previously discovered in [17]. Here we include this algorithm for completion.

We first construct a partial FDD from the first rule. A *partial FDD* is a diagram that has all the properties of an FDD except the completeness property.

The partial FDD constructed from a single rule contains only the decision path that defines the rule. Suppose from the first i rules we have constructed a partial FDD, whose root is v (v is labelled F_1, and suppose v has k outgoing edges $e_1, \cdots, e_k$). Let r_{i+1} be $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \rightarrow$ *decision*. Next we consider how to add rule r_{i+1} to this partial FDD.

At first, we examine whether we need to add another outgoing edge to v. If $S_1 - (I(e_1) \cup \cdots \cup I(e_k)) \neq \emptyset$, we need to add a new outgoing edge with label $S_1 - (I(e_1) \cup \cdots \cup I(e_k))$ to v because any packet whose F_1 field satisfies $S_1 - (I(e_1) \cdots \cup I(e_k))$ doesn't match any of the first i rules, but matches r_{i+1} if the packet satisfies $(F_2 \in S_2) \wedge \cdots \wedge (F_d \in S_d)$. This new edge points to the root of the partial FDD built from $(F_2 \in S_2) \wedge \cdots \wedge (F_d \in S_d) \rightarrow$ *decision*.

Second, we compare S_1 and $I(e_j)$ for each j where $1 \leq j \leq k$ 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 whose value of field F_0 is in set $I(e_j)$, it may match one of the first i rules, and it also may match rule r_{i+1}. So we add $(F_2 \in S_2) \wedge \cdots \wedge (F_d \in S_d) \rightarrow$ *decision* to the subgraph rooted at the node that e_j points to in a similar fashion.

3. $S_1 \cap I(e_j) \neq I(e_j)$ and $S_1 \cap I(e_j) \neq \emptyset$: 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 subgraph 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 following pseudocode of the construction algorithm, we use $e.t$ to denote the node that the edge e points to.

As an example, Figure 3.4 shows the partial FDD that we constructed the first rule, r_1, in Figure 1.1. The partial FDD that we get by appending the rule r_2 in Figure 1.1 to the partial FDD in 3.4 is in Figure 3 5. Figure 3.6 shows the FDD constructed from the sequence of rules in Figure 1.1 by the FDD construction algorithm in Figure 3.3.

```
FDD Construction Algorithm
Input  : A firewall $f$ of a sequence of rules $\langle r_1, \cdots, r_n \rangle$
Output : An FDD $f'$ such that $f$ and $f'$ are equivalent
Steps:
1. build a decision path with root $v$ from rule $r_1$;
2. for $i := 2$ to $n$ do Append( $v$, $r_i$ );

Append( $v$, $(F_m \in S_m) \wedge \cdots \wedge (F_d \in S_d) \rightarrow decision$ )
Input  : (1) Root $v$ of a partial FDD. Node $v$ is labelled $F_i$,
             and it has $k$ outgoing edges: $e_1, e_2, \cdots, e_k$
         (2) Rule $(F_m \in S_m) \wedge \cdots \wedge (F_d \in S_d) \rightarrow decision$
Output: The partial FDD with the above rule added
1. if ( $S_m - ( I(e_1) \cup \cdots \cup I(e_k) ) ) \neq \emptyset$ then
    (a) add an outgoing edge $e_{k+1}$ with label
        $S_m - (I(e_1) \cup \cdots \cup I(e_k))$ to $v$;
    (b) build a decision path from rule
        $(F_{m+1} \in S_{m+1}) \wedge \cdots \wedge (F_d \in S_d) \rightarrow decision$,
        and let $e_{k+1}$ point to its root;
2. if $m < d$ then
    for $j := 1$ to $k$ do
      if $I(e_j) \subseteq S_m$ then
        Append( $e_j.t$, $(F_{m+1} \in S_{m+1}) \wedge \cdots \wedge (F_d \in S_d)$
                                                    $\rightarrow decision$ );
      else if $I(e_j) \cap S_m \neq \emptyset$ then
        (a) $I(e_j) := I(e_j) - S_m$;
        (b) add one outgoing edge $e$ to $v$,
            and label $e$ with $I(e_j) \cap S_m$;
        (c) replicate the graph rooted at $e_j.t$,
            and let $e$ points to the replicated graph;
        (d) Append( $e.t$, $(F_{m+1} \in S_{m+1}) \wedge \cdots \wedge (F_d \in S_d)$
                                                    $\rightarrow decision$ );
```

Figure 3.3: FDD Construction Algorithm

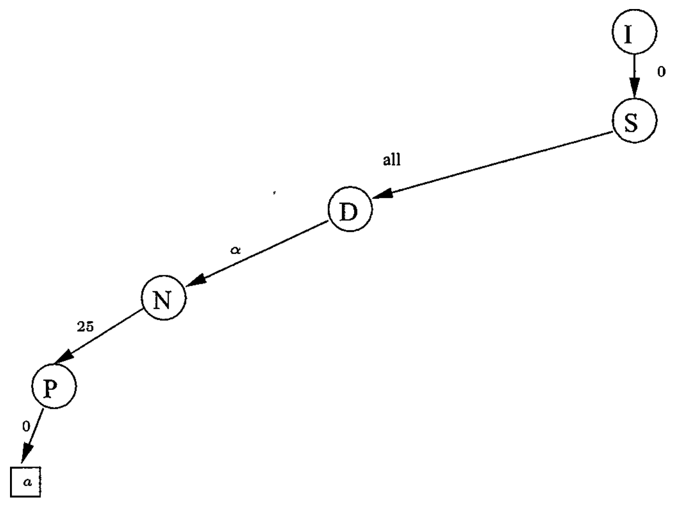

Figure 3.4: A partial FDD from rule r_1 in Figure 1.1

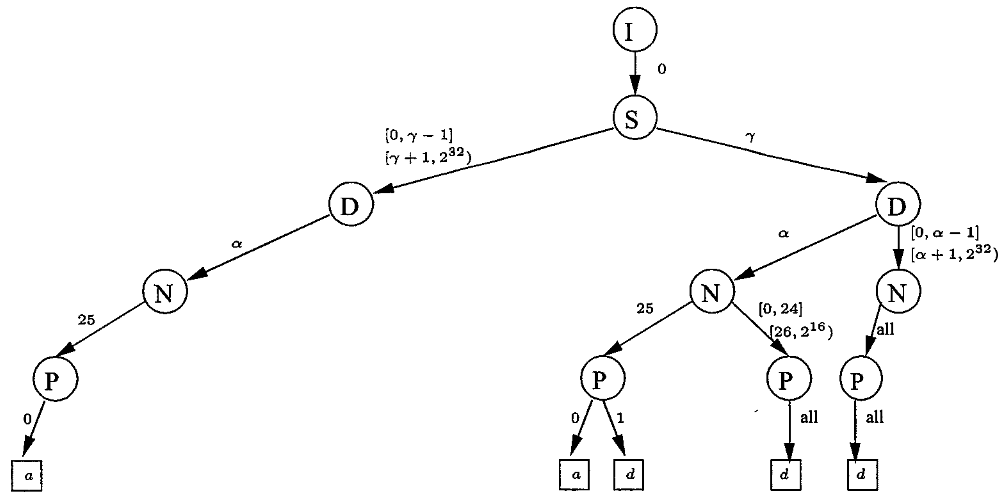

Figure 3.5: A partial FDD from rule r_1 and r_2 in Figure 1.1

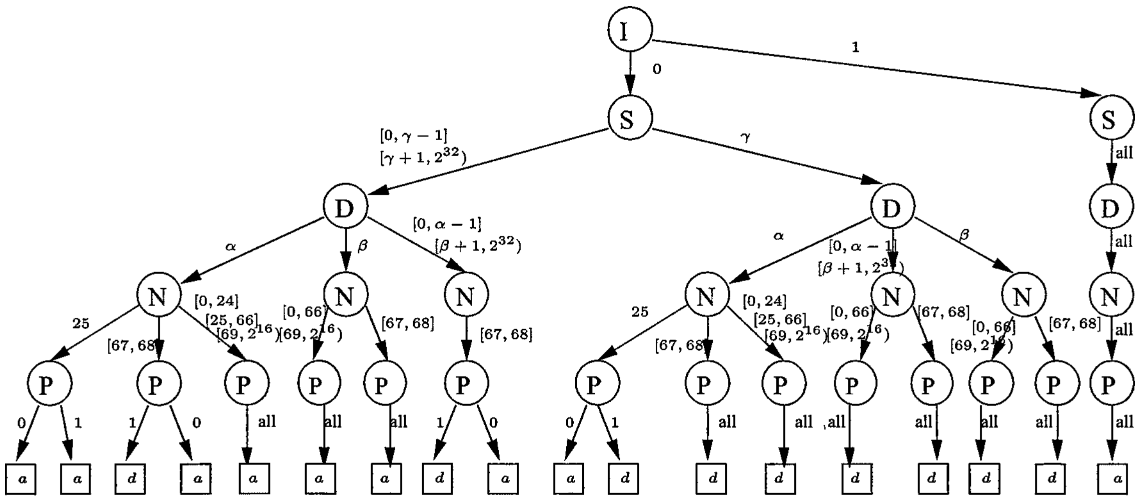

Figure 3.6: A constructed FDD from the firewall in Figure 1.1

Chapter 4

FIREWALL TESTING

From the previous chapter, we know that by applying the FDD construction algorithm in Figure 3.3 to a firewall of a sequence of rules, we get an equivalent FDD. In this Chapter, we consider how to answer firewall testing queries using an FDD.

Each query can be represented as $(F_1 \in Q_1) \wedge (F_2 \in Q_2) \wedge \cdots \wedge (F_d \in Q_d) \rightarrow$ *decision*, where Q_i is either a question mark "?" or a subset of $D(F_i)$. If some values of the field F_i will be part of the query result, we let Q_i be a question mark "?"; if the values of the field F_i have been specified by the query, we let Q_i be the specified value. For example, the query "which machines in the private network can receive BOOTP packets from the outside Internet?" can be represented as $(I \in \{0\}) \wedge (S \in \{all\}) \wedge (D \in ?) \wedge (N \in \{67, 68\}) \wedge (P \in \{1\}) \rightarrow a$.

A query $(F_1 \in Q_1) \wedge (F_2 \in Q_2) \wedge \cdots \wedge (F_d \in Q_d) \rightarrow$ *decision overlaps* with a rule $(F_1 \in S_1) \wedge (F_2 \in S_2) \wedge \cdots \wedge (F_d \in S_d) \rightarrow$ *decision'* iff the following two conditions hold:

1. for each i, $1 \leq i \leq d$, either $Q_i = ?$ or $Q_i \cap S(i) \neq \emptyset$ holds;

2. *decision* = *decision'*.

If a query $(F_1 \in Q_1) \wedge (F_2 \in Q_2) \wedge \cdots \wedge (F_d \in Q_d) \rightarrow$ *decision* with k $(1 \leq k \leq d)$ question marks, assuming $Q_{i_j} = ?$ for $1 \leq j \leq k$ and $i_1 < i_2 < \cdots < i_k$, overlaps with a rule $(F_1 \in S_1) \wedge (F_2 \in S_2) \wedge \cdots \wedge (F_d \in S_d) \rightarrow$ *decision'* defined by a decision path in an FDD, then the k-tuple $(S_{i_1}, S_{i_2}, \cdots, S_{i_k})$ is part of the answer to the query.

For example, the query "which machines in the private network can receive BOOTP packets from the outside Internet?" represented as $(I \in \{0\}) \wedge (S \in \{all\}) \wedge (D \in ?) \wedge (N \in \{67, 68\}) \wedge (P \in \{1\}) \rightarrow a$, overlaps with the rule $(I \in \{0\}) \wedge (S \in \{[0, \gamma - 1] \cup [\gamma + 1, 2^{32}]\}) \wedge (D \in \{\beta\}) \wedge (N \in [67, 68]) \wedge (P \in all) \rightarrow a$ defined by a decision path in the FDD in Figure 3.6, then the 1-tuple (β) is part of the answer to the query, i.e., the machine β in the private network can receive BOOTP packets from the outside Internet.

Given an FDD f and a query with k question marks, the set of tuples defined by all the rules in S_f that overlap with the query

```
Firewall Testing Algorithm
input  : (1) An FDD with root $v$ that we get from the FDD
             construction algorithm,
         (2) A query $(F_1 \in Q_1) \wedge (F_2 \in Q_2) \wedge \cdots \wedge (F_d \in Q_d)$
                 $\rightarrow decision$
output: Query result
1. $S := \emptyset$;
2. Query( $v$, $(F_1 \in Q_1) \wedge (F_2 \in Q_2) \wedge \cdots \wedge (F_d \in Q_d)$
                 $\rightarrow decision$ );
3. return($S$);

Query( $v$, $(F_i \in Q_i) \wedge (F_{i+1} \in Q_{i+1}) \wedge \cdots \wedge (F_d \in Q_d)$
                 $\rightarrow decision$ )
/*Let $E(v) = \{e_1, \cdots, e_m\}$.*/
/*Suppose $v$ is labelled $F_k$, where $i \leq k \leq d$.*/
for $j := 1$ to $m$ do
  if $Q_i = ?$ or $I(e_j) \cap Q_k \neq \emptyset$ then
    if $k < d$ then
      Query( $e_j.t$, $(F_{k+1} \in Q_{k+1}) \wedge \cdots \wedge (F_d \in Q_d)$ )
    else if $F(e_j.t) = decision$ then
      (1) Let $r$ be the rule defined by the decision path
          containing $e_j$;
      (2) Add the tuple defined by $r$ to $S$;
```

Figure 4.1: Firewall Testing Algorithm

Chapter 5

EXPERIMENTAL RESULTS

In this thesis, we presented a firewall testing algorithm for testing firewalls based on its specification. In this chapter, we mainly evaluate the efficiency of this firewall testing algorithm, which is measured by the average time for processing a firewall testing query.

In the absence of publicly available firewalls, we create synthetic firewalls at random. Each rule has the following five fields: interface, source IP address, destination IP address, destination port number and protocol type.

The programs are implemented in SUN Java JDK 1.4. The experiments were carried out on a SunBlade 2000 machine running Solaris 9 with 1Ghz CPU and 1 GB memory. Figure 5.1 shows the average execution time for processing a firewall testing query.

From this figure, we see that our firewall testing algorithm is both efficient and scalable. It takes less than 4 seconds to give answer to a firewall testing query even if the firewall has 5000 rules. In fact, it is very unlikely that a firewall can have this many rules. Clearly the efficiency of our firewall testing algorithm is more than enough to be used in practice. The intuitive explanation about the efficiency of our firewall testing algorithm is that processing of a query does not need to go through all the rules due to the constructed firewall decision diagram. Our firewall testing algorithm is scalable in terms of the number of rules. The running time of our firewall testing algorithm does not go exponentially with the number of rules.

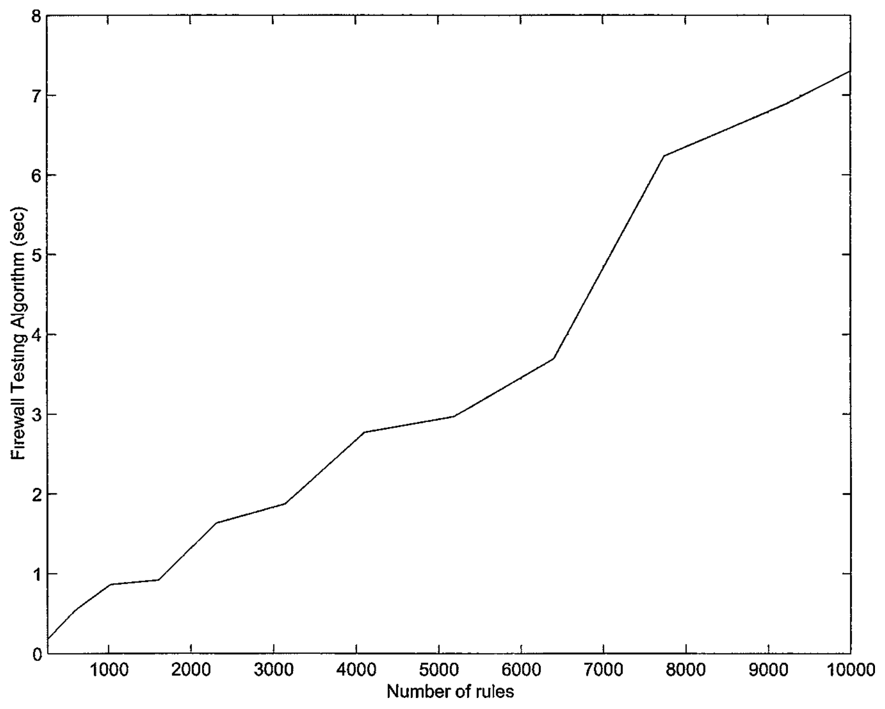

Figure 5.1: Experimental Results

Chapter 6

CONCLUDING REMARKS

Serving as the first line of defense against malicious attacks and unauthorized traffic, firewalls are important in securing the private network of most businesses, institutions, and even home networks. How to test the correctness of a firewall becomes an important problem.

In this thesis, we propose the method of specification based firewall testing using the data structure of Firewall Decision Diagrams. We presented a firewall testing algorithm for processing firewall test queries that generated from the specification of a firewall. The experimental results show that our firewall testing algorithm is very efficient.

What distinguishes our testing method and previous firewall testing methods is that we do not inject packets. Testing a firewall by injecting packets is very inefficient and hard to deploy. By our method, any aspect of a firewall specification can be easily tested in a few seconds. Both time and human effort involved is kept minimum. We consider our method of testing firewalls a new progress in the area of firewall testing.

Bibliography

[1] E. Al-Shaer and H. Hamed. Firewall policy advisor for anomaly detection and rule editing. In *IEEE/IFIP Integrated Management IM'2003*, March 2003.

[2] E. Al-Shaer and H. Hamed. Management and translation of filtering security policies. In *IEEE International Conference on Communications*, May 2003.

[3] E. Al-Shaer and H. Hamed. Discovery of policy anomalies in distributed firewalls. In *IEEE INFOCOM'04*, March 2004.

[4] A. W. Alain Mayer and E. Ziskind. Fang: A firewall analysis engine. In *Proc. of IEEE Symp. on Security and Privacy*, pages 177–187, 2000.

[5] 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.

[6] 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.

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

[8] P. Eronen and J. Zitting. An expert system for analyzing firewall rules. In *Proc. of the 6th Nordic Workshop on Secure IT Systems (NordSec 2001)*, pages 100–107, 2001.

[9] D. Farmer and W. Venema. Improving the security of your site by breaking into it. *http://www.alw.nih.gov/Security/Docs/admin-guide-to-cracking.101.html*, 1993.

[10] M. Freiss. *Protecting Networks with SATAN.* O'Reilly & Associates, Inc., 1998.

[11] M. G. Gouda and X.-Y. A. Liu. Firewall design: consistency, completeness and compactness. In *Proc. of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS'04)*, March 2004.

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

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

[14] S. Hazelhurst, A. Attar, and R. Sinnappan. Algorithms for improving the dependability of firewall and filter rule lists. In *Proc. of the International Conference on Dependable Systems and Networks (DSN'00)*, pages 576–585, 2000.

[15] J. Jürjens and G. Wimmel. Specification-based testing of firewalls. In A. Ershov, editor, *Proc. of the 4th International Conference Perspectives of System Informatics (PSI'01)*, LNCS. Springer.

[16] P. Krishnan and D. Hartley. Using model checking to test a firewall: a case study. In *Proc. of 28th Euromicro Conference*, pages 284–291, 2002.

[17] A. X. Liu and M. G. Gouda. Diverse firewall design. In *Proc. of the International Conference on Dependable Systems and Networks (DSN'04)*, June 2004.

[18] J. D. Moffett and M. S. Sloman. Policy conflict analysis in distributed system management. *Journal of Organizational Computing*, 4(1):1–22, 1994.

[19] Nessus. http://www.nessus org/. March 2004.

APPENDICES

APPENDIX 1

PROGRAMMING SOURCE CODE OF SPECIFICATION BASED FIREWALL TESING

The following material is the Java source code of the implementation of Specification Based Firewall Testing. The total number of the lines of code is about 1000.

1. Interval.java
2. Edge.java
3. Node.java
4. Rule.java
5. Sequence.java
6. testing.java

```
package firewalltesting;

/**
 * <p>Title: Specification Based Firewall Testing</p>
 * <p>Description: </p>
 * <p>Copyright: Huibo Ma, Copyright (c) 2004</p>
 * <p>Company: </p>
 * @author Huibo Ma
 * @version 1.0
 */

import java.util.*;

public class Interval {
  public int left, right;

  public Interval(int a, int b) {
    left = a;
    right = b;
  }

  public Interval( Interval S ) {
    left = S.left;
    right = S.right;
  }

  public ArrayList minus( Interval S ){
    //Does not affect this.left and  this.right
    ArrayList inArray = new ArrayList();

    if( (left <= S.right) && (S.left <= right) ){
      //Otherwise they don't overlap at all
      if( left < S.left)
        inArray.add( new Interval(left, S.left-1) );
    if( S.right < right )
      inArray.add( new Interval(S.right+1, right) );
   } else inArray.add( new Interval(left, right) );

   return inArray;
  }

  public ArrayList minus( Edge e ){ //Does not affect this.left and this.right
   ArrayList result = new ArrayList();
   ArrayList temp = new ArrayList();
   result.add( new Interval(left, right) );

   for( int i=0; i < e.size(); i++){
    for( int j=0; j < result.size(); j++){
      temp.addAll( ((Interval)(result.get(j))).minus( (Interval)( e.get(i) ) ) );
    }
    result.clear();
    result.addAll( temp );
    temp.clear();
   }

   return result;
  }

  public boolean overlap( Interval S ){
   return (left <= S.right) && (S.left <= right);
  }

  public boolean isSubsetOf( Interval S ){
   return (S.left <= left) && (right <= S.right);
  }

  public Interval intersection ( Interval S ){
   if( !overlap( S ) ) return null;
   else return new Interval( S.left > left? S.left:left, S.right > right?
right:S.right );
  }

  public boolean has( int x ){
   return (left<=x) && (x<=right);
```

```
 }

 public void print(){
  System.out.print("(" + left + "," + right + "), ");
 }

 public int getLeft(){
  return left;
 }

 public void setLeft(int a){
  left = a;
 }

 public int getRight(){
  return right;
 }

 public void setRight(int b){
  right=b;
 }
}
```

```
package firewalltesting;

/**
 * <p>Title: Specification Based Firewall Testing</p>
 * <p>Description: </p>
 * <p>Copyright: Huibo Ma, Copyright (c) 2004</p>
 * <p>Company: </p>
 * @author Huibo Ma
 * @version 1.0
 */

import java.util.*;

public class Edge extends ArrayList{
  private int fatherLabel;

  public Edge( int f ) { //The base class default constructor is called by
default
    fatherLabel = f;
  }

  public Edge( int f, Interval S ) {
    super.add( S );
    fatherLabel = f;
  }

  public Edge( int f, ArrayList a ) {
    super( a ); //Call the base class constructor
    fatherLabel = f;
  }

  public Edge( Edge e ) {
    //super( (ArrayList)e );
    for(int i=0; i<e.size(); i++){
      Interval S = new Interval( (Interval)(e.get(i)) );
    super.add( S );
  }
  fatherLabel = e.fatherLabel;
}

public void add( Interval S ){
  super.add( S );
}

public void add( int left, int right ){
  super.add( new Interval( left, right ) );
}

public int getFatherLabel(){
  return fatherLabel;
}

public void minus( Interval S ){ //This edge object is affected!
  ArrayList result = new ArrayList();

  for( int i=0; i < size(); i++)
    result.addAll( ((Interval)(get(i))).minus( S ) );

  this.clear();
  this.addAll(result);
}

public boolean overlap( Interval S ){
  for( int i=0; i < size(); i++ )
    if( S.overlap( (Interval)(get(i)) ) ) return true;

  return false;
}

public boolean isSubsetOf( Interval S ){
  for( int i=0; i < size(); i++ )
    if( !((Interval)(get(i))).isSubsetOf( S ) ) return false;

  return true;
}
```

```
  public ArrayList intersection( Interval S ){
    ArrayList result = new ArrayList();
    for( int i=0; i < size(); i++ ){
      Interval temp = S.intersection( (Interval)(get(i)) ); //If they don't overlap,
S.intersection returns null
      if( temp != null ) result.add( temp );
    }
    return result;
  }

  public ArrayList intersection( Edge e ){
    ArrayList result = new ArrayList();
    for( int i=0; i < e.size(); i++ ){
      ArrayList temp = this.intersection( (Interval)(e.get(i)) ); //If they don't
overlap, S.intersection returns null
      if( temp.size() > 0 ) result.addAll( temp );
    }
    return result;
  }

  public boolean has( int x ){

    for( int i=0; i<size(); i++ )
      if( ((Interval)get(i)).has( x ) ) return true;

    return false;
  }

  public void print(){
    System.out.print( fatherLabel + ":[");
    for( int i=0; i<size(); i++ )
      ((Interval)(get(i))).print();
    System.out.print("] && ");
  }
}
```

```
package firewalltesting;

/**
 * <p>Title: Specification Based Firewall Testing</p>
 * <p>Description: </p>
 * <p>Copyright: Huibo Ma, Copyright (c) 2004</p>
 * <p>Company: </p>
 * @author Huibo Ma
 * @version 1.0
 */

import java.util.*;

public class Node {
  private Edge inEdge;
  private int label;
  private ArrayList children;
  private boolean isLeaf;

  public Node(Edge e, int nodeLabel, boolean iamLeaf) {

    if (e == null)
      inEdge = new Edge( -1);
    else
      inEdge = new Edge(e);

    label = nodeLabel;
    isLeaf = iamLeaf;
    children = new ArrayList();
  }

  public Edge getInEdge() {
    return inEdge;
  }

  public Interval getInterval(int i) {
```

```
  return (Interval) (inEdge.get(i));
}

public void setInEdge(Edge e) {
  inEdge = e;
}

public int getLabel() {
  return label;
}

public void setLabel(int i) {
  label = i;
}

public boolean isLeaf() {
  return isLeaf;
}

public void addChild(Node son) {
  children.add(son);
}

public void addChild(int i, Node son) {
  children.add(i, son);
}

public Node getChild(int i) {
  return (Node) (children.get(i));
}

public ArrayList getAllChild() {
  return children;
}

public int getChildrenNum() {
  return children.size();
}

public Node deepCopy() {
  Node dest = new Node(inEdge, label, isLeaf());
  if (!isLeaf())
    for (int i = 0; i < children.size(); i++)
      dest.addChild(getChild(i).deepCopy());

  return dest;
 }

 public void printInEdge() {
  inEdge.print();
 }

 public void appendAtomicRule(Rule r, int beginPos) {
  if (label != r.getLabel(beginPos))
    System.out.println("Label = " + label + ", but r.getLabel( beginPos ) = "
+
                   r.getLabel(beginPos));
  else {
    Interval Si = new Interval( (Interval) (r.get(beginPos).get(0)));

    //Calculate the label of the edge that lead to a chain
    Edge oldEdges = new Edge(label);
    for (int i = 0; i < getChildrenNum(); i++)
      oldEdges.addAll(getChild(i).getInEdge());
    Edge newEdge = new Edge(label, Si.minus(oldEdges));

    //Check every existing edge against Si to see whether we need to break
existing edges
    if (beginPos < r.size() - 1) {
      for (int j = 0; j < children.size(); j++) {
        Node n = getChild(j);
        Edge Ei = n.getInEdge();

        if (Ei.isSubsetOf(Si))
          n.appendAtomicRule(r, beginPos + 1);
        else if (Ei.overlap(Si)) {
          ArrayList overlap = Ei.intersection(Si);
          Node copy = n.deepCopy();
          Edge e = copy.getInEdge();
          e.clear();
          e.addAll(overlap);
          addChild(copy);
          Ei.minus(Si);
          copy.appendAtomicRule(r, beginPos + 1);
        }
      }
    }

    //Build the chain
    if (newEdge.size() != 0) {
      Node newSon = r.buildChain(beginPos + 1);
      newSon.setInEdge(newEdge);
      addChild(newSon);
    }

  }
 }

 public void appendAtomicRuleForDownward(Rule r, int beginPos) {
   if (label != r.getLabel(beginPos))
     System.out.println("Label = " + label + ", but r.getLabel( beginPos ) = "
+
                   r.getLabel(beginPos));
   else {
     Interval Si = new Interval( (Interval) (r.get(beginPos).get(0)));

     //Check every existing edge against Si to see whether we need to break
existing edges
     if (beginPos < r.size() - 1) {
       for (int j = 0; j < this.getChildrenNum(); j++) {
         Node n = getChild(j);
         Edge Ei = n.getInEdge();

         if (Ei.isSubsetOf(Si))
           n.appendAtomicRuleForDownward(r, beginPos + 1);
         else if (Ei.overlap(Si)) {
           ArrayList overlap = Ei.intersection(Si);
           Node copy = n.deepCopy();
           Edge e = copy.getInEdge();
           e.clear();
           e.addAll(overlap);
           addChild(copy);
           Ei.minus(Si);
           copy.appendAtomicRuleForDownward(r, beginPos + 1);
         }
       }
     }
     else {
       for (int j = 0; j < this.getChildrenNum(); j++) {
         Node n = getChild(j);
         Edge Ei = n.getInEdge();
         //Ei.print();
         //Si.print();
         Ei.minus(Si);
         if (Ei.isEmpty()) {
           children.remove(j);
           j--;
         }
       }

       this.addChild(new Node(new Edge(this.getLabel(), Si), r.getDecision(),
true));
     }
   }
 }

 public Sequence deriveRules() {
   Rule rule = new Rule();
   Sequence result = new Sequence();

   derive(this, rule, result);

   return result;
 }

 private void derive(Node v, Rule rule, Sequence result) {
   if (v.isLeaf()) {
    rule.setDecision(v.getLabel());
    result.add(new Rule(rule));
    rule.setDecision( -1);
   }
   else {
    for (int i = 0; i < v.getChildrenNum(); i++) {
     Node son = v.getChild(i);
     rule.add(new Edge(son.getInEdge()));
     derive(son, rule, result);
     rule.removeLastEdge();
    }
   }
 }

 public void sort() {
   /*   System.out.print("Before Sorting:");
     for( int i=0; i<getChildrenNum(); i++ ){
      System.out.print("Edge " + i);
      getChild(i).getInEdge().print();
     }
     System.out.println();
    */
   for (int i = 1; i < getChildrenNum(); i++) {
    int j = i;
    while (j > 0) {
     Node nj = getChild(j);
     Interval Sj = nj.getInterval(0);
     int leftj = Sj.getLeft();

     Node nj1 = getChild(j - 1);
     Interval Sj1 = nj1.getInterval(0);
     int leftj1 = Sj1.getLeft();

     if (leftj < leftj1) {
      children.remove(j);
      children.add(j - 1, nj);
     }
     else
      break;
     j--;
    }
   }
   /*
```

```
        System.out.print("After Sorting:");
        for( int i=0; i<getChildrenNum(); i++ ){
          System.out.print("Edge " + i);
          getChild(i).getInEdge().print();
        }
        System.out.println();
      */
//    for( int i=0; i<children.size(); i++ ){
//      Node son = (Node)(children.get(i));
//      son.sort();
//    }
  }

  public void simplify() {
    /*    System.out.print("Before Simplifying:");
        for( int i=0; i<getChildrenNum(); i++ ){
          System.out.print("Edge " + i);
          getChild(i).getInEdge().print();
        }
        System.out.println();
      */
    int m = children.size();
    for (int i = 0; i < m; i++) {
      Node son = getChild(i);
      Edge e = son.getInEdge();
      while (e.size() > 1) {
        Interval S = (Interval) (e.get(1));
        Node brother = son.deepCopy();
        brother.setInEdge(new Edge(label, S));
        children.add(brother);
        e.remove(1);
      }
    }
    /*
        System.out.print("After Simplifying:");
        for( int i=0; i<getChildrenNum(); i++ ){
          System.out.print("Edge " + i);
          getChild(i).getInEdge().print();
        }
        System.out.println();
    */
//   for( int i=0; i<getChildrenNum(); i++ ){
//     Node son = getChild(i);
//     son.simplify();
//   }
  }

  public void print() {
    System.out.print("Node:");
    if (inEdge != null)
      inEdge.print();

    System.out.println();
    for (int i = 0; i < getChildrenNum(); i++) {
      getChild(i).print();
    }
  }

  /*For Firewall Testing*/
 /* public Sequence FirewallTest(){
    Rule path = new Rule();
    Rule query = new Rule();

    Sequence result = new Sequence();

    derive( this, path, result);

    return result;
  }*/



}
```

```
package firewalltesting;

/**
 * <p>Title: Specification Based Firewall Testing</p>
 * <p>Description: </p>
 * <p>Copyright: Huibo Ma, Copyright (c) 2004</p>
 * <p>Company: </p>
 * @author Huibo Ma
 * @version 1.0
 */

import java.util.*;

public class Rule {
  private ArrayList edges; //data is an array of edges
  private int decision;

  public Rule() {
    edges = new ArrayList();
    decision = -1;  //Indicate this decision is not initialized
  }

  public Rule( Rule r ){
    edges = new ArrayList( r.getEdgeList() );
    decision = r.decision;
  }

  public ArrayList getEdgeList(){
    return edges;
  }

  public void setEdgeList( ArrayList a ){
    edges = a;
  }

  public int getDecision(){
```

```
  return( decision );
}

public void setDecision( int i ){
  decision = i;
}

public Edge get( int i ){
  return( (Edge)(edges.get(i)) );
}

public int getLabel( int i ){
  return get(i).getFatherLabel();
}

public int getLastLabel(){
  return getLabel( size() - 1 );
}

public void add( Edge e ){
  edges.add( e );
}

public void removeLastEdge(){
  edges.remove( edges.size() - 1 );
}

public int size(){
  return edges.size();
}

public Rule combineRulesCopy( Rule r, int beginPos ){
  Rule resultRule = new Rule();
  resultRule.setDecision( r.getDecision() );

  for( int i=0; i < this.size(); i++ ){
    resultRule.add( new Edge( this.get( i ) ) );
  }

  for( int i=beginPos; i < r.size(); i++ ){
   resultRule.add( new Edge( r.get( i ) ) );
  }

  return resultRule;
 }

 public Sequence simplify(){
  Sequence atomicSeq = new Sequence();
  for( int i=0; i<this.get(0).size(); i++)
   for( int j=0; j<this.get(1).size(); j++)
    for( int k=0; k<this.get(2).size(); k++)
     for( int m=0; m<this.get(3).size(); m++)
      for( int n=0; n<this.get(4).size(); n++){
       Rule r = new Rule();
       r.add(new Edge(0, (Interval) (this.get(0).get(i))));
       r.add(new Edge(1, (Interval) (this.get(1).get(j))));
       r.add(new Edge(2, (Interval) (this.get(2).get(k))));
       r.add(new Edge(3, (Interval) (this.get(3).get(m))));
       r.add(new Edge(4, (Interval) (this.get(4).get(n))));
       r.setDecision(this.getDecision());
       atomicSeq.add(r);
      }
  return atomicSeq;
 }

 public void print(){ //For debugging
  for( int i=0; i < size(); i++ )
   get(i).print();

  System.out.println( "->" + decision );
 }

 public Node buildChain( int begin ){
  Node root, father, son;

  if( begin == size() ){ //This means to create a terminal node
   root = new Node( get( size()-1 ), decision, true );
  }else{
   //root = new Node( null, begin, false );
   root = new Node( null, get(begin).getFatherLabel(), false );
```

```
      father = root;
      for (int i = begin; i < size() - 1; i++) {
        son = new Node(get(i), get(i+1).getFatherLabel(), false);
        father.addChild(son);
        father = son;
      }
      son = new Node(get(size() - 1), decision, true);
      father.addChild(son);
    }

    /*//test
    if( !root.isLeaf() ){
      son = (Node) (root.getChild(0));
      System.out.print("Chain: ");
      while (!son.isLeaf()) {
        son.printInEdge();
        son = (Node) (son.getChild(0));
      }
      son.printInEdge();
      System.out.println( "-->" + son.getLabel() );
    }
    else{
      root.printInEdge();
      System.out.println( "-->" + root.getLabel() );
    }*/

    return root;
  }

  public int resolve( int[] packet ){
    boolean match = true;

    for( int i=0; i<size(); i++ ){
      Edge e = get(i);
      if( !e.has( packet[e.getFatherLabel()] ) ) match = false;
    }

    return match? decision:-1;
  }
}
```

```
package firewalltesting;

/**
 * <p>Title: Specification Based Firewall Testing</p>
 * <p>Description: </p>
 * <p>Copyright: Huibo Ma, Copyright (c) 2004</p>
 * <p>Company: </p>
 * @author Huibo Ma
 * @version 1.0
 */

import java.util.*;

public class Sequence {
  private ArrayList rules;

  private int ruleDim=5; //the number of variables in each rule
  private int[] range = new int[ruleDim + 1];

  public Sequence(){
    rules = new ArrayList();

    range[0] = 2; //Interface: 0..1
    range[1] = 65536; //Souce IP Address: 0..2^{16}-1. Assuming an IP
address uses two bytes. 2^{16}=65536
    range[2] = 65536; //Destination IP Address: 0..2^{16}-1
    range[3] = 256; //Port Number: 0..2^{8}-1. Byte. MAX_VALUE =
2^{7}-1
    range[4] = 8; //Protocol type: 0..7
    range[5] = 2; //Decision: 0..1
  }

  public Sequence(int demo){
    rules = new ArrayList();

    range[0] = 2; //Interface: 0..1
  range[1] = 10; //Souce IP Address: 0..2^{16}-1. Assuming an IP address
uses two bytes. 2^{16}=65536
  range[2] = 10; //Destination IP Address: 0..2^{16}-1
  range[3] = 10; //Port Number: 0..2^{8}-1. Byte. MAX_VALUE = 2^{7}-
1
  range[4] = 10; //Protocol type: 0..7
  range[5] = 2; //Decision: 0..1*/
 }


 public Edge createRandomEdge( int label, int range ){
  Random ranVal = new Random();
  int a = ranVal.nextInt( range );
  int b = ranVal.nextInt( range );

  if( a < b )
   return( new Edge( label, new Interval( a, b ) ) );
  else
   return( new Edge( label, new Interval( b, a ) ) );
 }

 public Rule createQuery(){
  Rule query = new Rule();
  Random ranVal = new Random();

  query.add( createRandomEdge( 0, range[0] ) );
  query.add( createRandomEdge( 1, range[1] ) );
  query.add( createRandomEdge( 2, range[2] ) );
  query.add( createRandomEdge( 3, range[3] ) );
  query.add( createRandomEdge( 4, range[4] ) );

  query.setDecision( ranVal.nextInt(2) );

  return( query );
 }

 public Rule createQueryForDemo(){
  Rule query = new Rule();
  Random ranVal = new Random();
  int pos=ranVal.nextInt(5);

  if( pos == 0 ) query.add(  new Edge( 0, new Interval( 0, range[0]-1 ) ) );
  else{
    int a = ranVal.nextInt(1);
    query.add(new Edge(0, new Interval(a, a)));
  }

  if( pos == 1 ) query.add(  new Edge( 1, new Interval( 0, range[1]-1 ) ) );
  else query.add( createRandomEdge( 1, range[1] ) );

  if( pos == 2 ) query.add(  new Edge( 2, new Interval( 0, range[2]-1 ) ) );
  else query.add( createRandomEdge( 2, range[2] ) );

  if( pos == 3 ) query.add(  new Edge( 3, new Interval( 0, range[3]-1 ) ) );
  else query.add( createRandomEdge( 3, range[3] ) );

  if( pos == 4 ) query.add(  new Edge( 4, new Interval( 0, range[4]-1 ) ) );
  else query.add( createRandomEdge( 4, range[4] ) );

  query.setDecision( ranVal.nextInt(2) );

  return( query );
}

//totally random value for each rule
//ruleNum : the total number of rules in the sequence
public int CreateByRandom( int ruleNum ) {

  Random ranVal = new Random();
  Rule r;
  for(int i = 0; i < ruleNum-1; i++){
    r = new Rule();
    int left, right;

    //Interface
    left = ranVal.nextInt(range[0]);
    r.add( new Edge( 0, new Interval(left, left) ) );

    //Source IP
    left = ranVal.nextInt(range[1]);
    right = ranVal.nextInt(range[1]);
    r.add( new Edge(1, new Interval( left<right? left:right, left<right?
right:left) ) );

    //Destination IP
    left = ranVal.nextInt(range[2]);
    right = ranVal.nextInt(range[2]);
    r.add( new Edge(2, new Interval( left<right? left:right, left<right?
right:left) ) );

    //Port Number
    left = ranVal.nextInt(range[3]);
    right = ranVal.nextInt(range[3]);
    r.add( new Edge(3, new Interval( left<right? left:right, left<right?
right:left) ) );

    //Protocol Type
    left = ranVal.nextInt(range[4]);
    r.add( new Edge(4, new Interval( left, left ) ) );

    //Decision
    r.setDecision(ranVal.nextInt(range[5]));

    add(r);
   }

   //Add last rule
   r = new Rule();
   for( int j=0; j<=ruleDim-1; j++ ){
    r.add(new Edge(j, new Interval(0, range[j] - 1)));
   }
   r.setDecision(ranVal.nextInt(range[ruleDim]));
   add( r );

   return( ruleNum);
  }

  public int CreateRules( int A, int B, int C ){
   Rule base = new Rule();
```

```
Edge e;
Random ranVal = new Random();
int[] points;

e = new Edge( 0 );
e.add( 0, 0 );
//e.add( 3, 3 );
//e.add( 5, 5 );
e.add( 0, 7 );
base.add( e );

e = new Edge( 1 );
points = new int[2*(A-1)];
points[0] = 0;
for( int i=1; i<2*(A-1); i++ )
  points[i] = ranVal.nextInt(range[1]);
sort(2*(A-1), points);
for( int i=0; i<A-1; i++ ){
  e.add( points[2*i], points[2*i+1] );
}
e.add( 0, range[1] - 1 );
base.add( e );

e = new Edge( 2 );
points = new int[2*(B-1)];
points[0] = 0;
for( int i=1; i<2*(B-1); i++ )
  points[i] = ranVal.nextInt(range[2]);
sort(2*(B-1), points);
for( int i=0; i<B-1; i++ ){
  e.add( points[2*i], points[2*i+1] );
}
e.add( 0, range[2] - 1 );
base.add( e );

e = new Edge( 3 );
points = new int[2*(C-1)];
points[0] = 0;
for( int i=1; i<2*(C-1); i++ )
  points[i] = ranVal.nextInt(range[3]);
```

```
 sort(2*(C-1), points);
 for( int i=0; i<C-1; i++ ){
  e.add( points[2*i], points[2*i+1] );
 }
 e.add( 0, range[3] - 1 );
 base.add( e );

 e = new Edge( 4 );
 e.add( 0, 0 );
 //e.add( 4, 4 );
 //e.add( 6, 6 );
 e.add( 0, 7 );
 base.add( e );

 //base.print();
 for (int i = 0; i < base.get(0).size(); i++)
  for (int j = 0; j < base.get(1).size(); j++)
   for (int k = 0; k < base.get(2).size(); k++)
    for (int m = 0; m < base.get(3).size(); m++)
     for (int n = 0; n < base.get(4).size(); n++){
      Interval S0 = (Interval)(base.get(0).get(i));
      Interval S1 = (Interval)(base.get(1).get(j));
      Interval S2 = (Interval)(base.get(2).get(k));
      Interval S3 = (Interval)(base.get(3).get(m));
      Interval S4 = (Interval)(base.get(4).get(n));
      Rule r = new Rule();
      r.add( new Edge(0, S0) );
      r.add( new Edge(1, S1) );
      r.add( new Edge(2, S2) );
      r.add( new Edge(3, S3) );
      r.add( new Edge(4, S4) );
      r.setDecision( ranVal.nextInt(range[5]) );
      rules.add( r );
     }

 return 4*A*B*C;
}

public static void sort( int len, int[] array ){
 for( int i=1; i<len; i++ ){
    int j = i;
    int temp;
    while( j > 0 ){
      if( array[j] < array[j-1] ) {
        temp = array[j];
        array[j] = array[j-1];
        array[j-1] = temp;
      }
      else break;
      j--;
    }
  }
}

public Rule get( int i ){
  return( (Rule)(rules.get(i)) );
}

public void add( Rule r ){
  rules.add( r );
}

public int size(){
  return rules.size();
}

public void print(){ //For debugging
  for( int i=0; i<size(); i++ ){
    System.out.print( "Rule " + i + ":" );
    get(i).print();
  }
}

public boolean isEmpty(){
  return size() == 0;
}

public Node buildFDD(){
  if( isEmpty() ) return null;
```

```
    Node root = get(0).buildChain(0);
    for( int i=1; i<rules.size(); i++ )
      root.appendAtomicRule( get(i), 0 );

    return root;
  }

  public int resolve( int[] packet ){
    int result = -1;
    for( int i=0; i<size(); i++){
      result = get(i).resolve( packet );
      if( result != -1 ) return result;
    }
    return result;
  }

  public int getRange( int i ){
    if( (i>=0) && (i<=ruleDim) ) return this.range[ i ];
    else return -1;
  }

  public boolean isEquivalent( Sequence seq, int range0, int range1, int
range2, int range3, int range4 ){
    int[] packet = new int[ruleDim];
    for (int i = 0; i < range0; i++)
      for (int j = 0; j < range1; j++)
        for (int k = 0; k < range2; k++)
          for (int m = 0; m < range3; m++)
            for (int n = 0; n < range4; n++){
              packet[0] = i;
              packet[1] = j;
              packet[2] = k;
              packet[3] = m;
              packet[4] = n;
              if (resolve(packet) != seq.resolve(packet)) {
                System.out.println("Different! Packet = " + i + "," + j + "," + k +
"," + m + "," + n);
                return false;
              }
            }
  return true;
 }

 public long packetTesting( Sequence seq, int range0, int range1, int range2,
int range3, int range4 ){
  long sum=0;
  int[] packet = new int[ruleDim];
  for (int i = 0; i < range0; i++)
   for (int j = 0; j < range1; j++)
    for (int k = 0; k < range2; k++)
     for (int m = 0; m < range3; m++)
      for (int n = 0; n < range4; n++){
       packet[0] = i;
       packet[1] = j;
       packet[2] = k;
       packet[3] = m;
       packet[4] = n;
       if (resolve(packet) != seq.resolve(packet)) {
        sum++;
       }
      }
  return sum;
 }

 public void clear(){
  rules.clear();
 }

 public void CreateNiceRandom( int ruleNum ){
  Random ranVal = new Random();
  Rule r;
  for (int i = 0; i < ruleNum-1; i++) {
   r = new Rule();
   int left, right;

   for (int j = 0; j < 5; j++) {
    left = ranVal.nextInt(10);
    right = ranVal.nextInt(10);
    r.add(new Edge(j, new Interval(left < right ? left : right,
                   left < right ? right : left)));
    }

    //Decision
    r.setDecision(ranVal.nextInt(2));
    add(r);
   }

  r = new Rule();
  for (int j = 0; j < 5; j++)
     r.add(new Edge(j, new Interval(0, 9)));
  r.setDecision(ranVal.nextInt(2));
  add(r);
 }

 public void CreateForTest() {
  Rule r;

  r = new Rule();
  r.add( new Edge(0, new Interval( 20,50 ) ) );
  r.add( new Edge(1, new Interval( 35,65 ) ) );
  r.setDecision(0);
  this.add(r);

  r = new Rule();
  r.add( new Edge(0, new Interval( 10,60 ) ) );
  r.add( new Edge(1, new Interval( 15,45 ) ) );
  r.setDecision(1);
  this.add(r);

  r = new Rule();
  r.add( new Edge(0, new Interval( 30,40 ) ) );
  r.add( new Edge(1, new Interval( 25,55 ) ) );
  r.setDecision(0);
  this.add(r);

  r = new Rule();
  r.add( new Edge(0, new Interval( 1,100 ) ) );
  r.add( new Edge(1, new Interval( 1,100 ) ) );
  r.setDecision(1);
  this.add(r);
  }
}
package firewalltesting;

/**
 * <p>Title: Specification Based Firewall Testing</p>
 * <p>Description: </p>
 * <p>Copyright: Huibo Ma, Copyright (c) 2004</p>
 * <p>Company: </p>
 * @author Huibo Ma
 * @version 1.0
 */

import java.util.*;
import java.io.*;

public class testing {
  public testing() {
  }
  public static void main(String[] args) {
    //testing testing1 = new testing();
    firewallTestingForDemo();
    //firewallTesting();
  }

  private static void QueryProcess( Node v, Rule testQuery, Rule
intersectPath, Sequence result){
    if( v.isLeaf() ){
      if( v.getLabel() == testQuery.getDecision() ){
        intersectPath.setDecision( v.getLabel() );
        result.add(new Rule(intersectPath));
      }
    }
    else{
      for( int i=0; i<v.getChildrenNum(); i++){
        Node son = v.getChild(i);
        ArrayList intersect = testQuery.get(v.getLabel()).intersection(new
Edge(son.getInEdge()));
        if( intersect.size() > 0){
          intersectPath.add(new Edge(v.getLabel(), intersect));
        QueryProcess(son, testQuery, intersectPath, result);
        intersectPath.removeLastEdge();
      }
    }
  }
}

private static void firewallTestingForDemo(){
  Sequence seq = new Sequence(1);

  int ruleNum = seq.CreateByRandom( 2 );

  //Create a query
  Rule query = seq.createQueryForDemo();

  //Construct FDD
  Node root = seq.buildFDD();

  //debug
  System.out.println("Rules Generated from the FDD:" );
  root.deriveRules().print();

  System.out.println("Query is:" );
  query.print();

  Sequence result = new Sequence();
  Rule intersectPath = new Rule();

  QueryProcess( root, query, intersectPath, result);

  //debug
  System.out.println("Result:" );
  result.print();
}


private static void firewallTesting(){
  Sequence seq = new Sequence();
  long start = 0;
  long end = 0;
```

```
   long sum =0;
   int repeat = 1000;

   PrintStream file;

   try{
     file = new PrintStream(new BufferedOutputStream(new
FileOutputStream("TestResult_x_x_16_repeat1000_real.txt")));

     for( int x=2; x<=15; x++ ){
       sum = 0;

       int ruleNum = seq.CreateRules(x, x, 16);
       Node root = seq.buildFDD();

       for( int i=0; i < repeat; i++ ){
         Rule query = seq.createQuery();
         Rule intersectPath = new Rule();
         Sequence result = new Sequence();

         start = System.currentTimeMillis();
         QueryProcess( root, query, intersectPath, result);
         end = System.currentTimeMillis();
         sum = sum + (end - start);

         seq.clear();
         System.gc();
       }

       float time = (float)(sum)/(float)repeat;
       file.println( ruleNum + ":   " + time );
       file.flush();
     }
     file.close();
   }catch(IOException e) {
     e.printStackTrace();
     System.err.println("Exception Happens!!! Heidi!!!");
   }
  }
```

```
}
```

VITA

Huibo Heidi Ma was born in Changchun, Jilin Province, China on February 15, 1974. After completing her studies at Jilin Shiyan High School, Jilin, in 1992, she entered the Computer Science Department of Jilin University, where the first Computer Science Department was set up in China. She received the degree of Bachelor of Science in Computer Science in 1996. During the following four years, she worked in software development industry in Beijing, China. In September, 2002, she entered The Graduate College of Texas State University-San Marcos and worked on her degree of Master of Science.

Permanent Address: USA

This thesis was typeset with LaTeX 2ε[1] by the author.

[1]LaTeX 2ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society.