Firewall Queries

Alex X. Liu[1], Mohamed G. Gouda[1], Huibo H. Ma[2], Anne HH. Ngu[2]

[1] Department of Computer Sciences, The University of Texas at Austin,
Austin, Texas 78712-0233, U.S.A.
{alex, gouda}@cs.utexas.edu
[2] Department of Computer Science, Texas State University,
San Marcos, Texas 78666-4616, U.S.A.
{hm1034, angu}@txstate.edu

Abstract. Firewalls are crucial elements in network security, and have been widely deployed in most businesses and institutions for securing 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 based on a sequence of rules. Because a firewall may have a large number of rules and the rules often conflict, understanding and analyzing the function of a firewall have been known to be notoriously difficult. An effective way to assist humans in understanding and analyzing the function of a firewall is by issuing firewall queries. An example of a firewall query is "Which computers in the private network can receive packets from a known malicious host in the outside Internet?". Two problems need to be solved in order to make firewall queries practically useful: how to describe a firewall query and how to process a firewall query. In this paper, we first introduce a simple and effective SQL-like query language, called the Structured Firewall Query Language (SFQL), for describing firewall queries. Second, we present a theorem, called the Firewall Query Theorem, as a foundation for developing firewall query processing algorithms. Third, we present an efficient firewall query processing algorithm, which uses firewall decision trees as its core data structure. Experimental results show that our firewall query processing algorithm is very efficient: it takes less than 10 milliseconds to process a query over a firewall that has up to 10,000 rules.
Keywords: Network Security, Firewall Queries, Firewalls

1 Introduction

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. For the sake of brevity, we assume that each packet has a field that contains the identification of the network interface on which a packet arrives. The $\langle decision \rangle$ of a rule can be *accept*, or *discard*, or a combination of these decisions with other options such as the logging option. For simplicity, we assume that the $\langle decision \rangle$ in a rule is either *accept* or *discard*.

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 often conflict. Two rules in a firewall *conflict* iff they have different decisions and 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.

The function (i.e., behavior) of a firewall is specified in its configuration, which consists of a sequence of rules. The configuration of a firewall is the most important component in achieving the security and functionality of the firewall [24]. However, most firewalls on the Internet are poorly configured, as witnessed by the success of recent worms and viruses like Blaster [6] and Sapphire [7], which could easily be blocked by a well-configured firewall [26]. It has been observed that most firewall security breaches are caused by configuration errors [5]. An error in a firewall configuration means that some illegitimate packets are identified as being legitimate, or some legitimate packets are identified as being illegitimate. This will either allow unauthorized access from the outside Internet to the private network, or disable some legitimate communication between the private network and the outside Internet. Neither case is desirable. Clearly, a firewall configuration should be well understood and analyzed before being deployed.

However, due to the large number of rules in a firewall and the large number of conflicts among rules, understanding and analyzing the function of a firewall have been known to be notoriously difficult [21]. The implication of any rule in a firewall cannot be understood without examining all the rules listed above that rule. There are other factors that contribute to the difficulties in understanding and analyzing firewalls. For example, a corporate firewall often consists of rules that are written by different administrators at different times and for different reasons. It is difficult for a new firewall administrator to understand the implication of each rule that is not written by herself.

An effective way to assist humans in understanding and analyzing firewalls is by issuing firewall queries. Firewall queries are questions concerning the function

of a firewall. Examples of firewall queries are "Which computers in the outside Internet cannot send emails to the mail server in a private network?" and "Which computers in the private network can receive BOOTP[1] packets from the outside Internet?". Figuring out answers to these firewall queries is of tremendous help for a firewall administrator to understand and analyze the function of the firewall. For example, assuming the specification of a firewall requires that all computers in the outside Internet, except a known malicious host, are able to send emails to the mail server in the private network, a firewall administrator can test whether the firewall satisfies this requirement by issuing a firewall query "Which computers in the outside Internet cannot send emails to the mail server in the private network?". If the answer to this query contains exactly the known malicious host, then the firewall administrator is assured that the firewall does satisfy this requirement. Otherwise the firewall administrator knows that the firewall fails to satisfy this requirement, and she needs to reconfigure the firewall. As another example, suppose that the specification of a firewall requires that any BOOTP packet from the outside Internet is to be blocked from entering the private network. To test whether the firewall satisfies this requirement, a firewall administrator can issue a firewall 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 the firewall administrator is assured that the firewall does satisfy this requirement. Otherwise the firewall administrator knows that the firewall fails to satisfy this requirement, and she needs to reconfigure the firewall.

Firewall queries are also useful in a variety of other scenarios, such as firewall maintenance and firewall debugging. For a firewall administrator, checking whether a firewall satisfies certain conditions is part of daily maintenance activity. For example, if the administrator detects that a computer in the private network is under attack, the firewall administrator can issue queries to check which other computers in the private network are also vulnerable to the same type of attacks. In the process of designing a firewall, the designer can issue some firewall queries to detect design errors by checking whether the answers to the queries are consistent with the firewall specification.

To make firewall queries practically useful, two problems need to be solved: how to describe a firewall query and how to process a firewall query. The second problem is technically difficult. Recall that the rules in a firewall are sensitive to the rule order and the rules often conflict. The naive solution is to enumerate every packet specified by a query and check the decision for each packet. Clearly, this solution is infeasible. For example, to process the query "Which computers in the outside Internet cannot send any packet to the private network?", this

[1] The Bootp protocol is used by workstations and other devices to obtain IP addresses and other information about the network configuration of a private network. Since there is no need to offer the service outside a private network, and it may offer useful information to hackers, usually Bootp packets are blocked from entering a private network.

naive solution needs to enumerate 2^{88} possible packet and check the decision of the firewall for each packet, which is infeasible.

In this paper, we present solutions to both problems. First, we introduce a simple and effective SQL-like query language, called the Structured Firewall Query Language (SFQL), for describing firewall queries. This language uses queries of the form "*select...from...where...*". Second, we present a theorem, called the Firewall Query Theorem, as the foundation for developing firewall query processing algorithms. Third, we present an efficient query processing algorithm that uses firewall decision trees as its core data structure. For a given firewall of a sequence of rules, we first construct an equivalent firewall decision tree by a construction algorithm. Then the firewall decision tree is used as the core data structure of this query processing algorithm for answering each firewall query. Experimental results show that our firewall query processing algorithm is very efficient: it takes less than 10 milliseconds to process a query over a firewall that has up to 10,000 rules. Clearly, our firewall query processing algorithm is fast enough in interacting with firewall administrators.

Note that firewalls that we consider in this paper are static firewalls, not stateful firewalls in which the function of a firewall changes dynamically as packets pass by. Also note that the queries of a firewall are intended primarily for the administrator of the firewall to use. For a firewall that protects a private network, neither normal users in the private network nor the outsiders of the private network are able to query the firewall. Since the focus of this paper is firewall configurations, in the rest of this paper, we use "firewall" to mean "firewall configuration" if not otherwise specified.

2 Related Work

There is little work that has been done on firewall queries. In [21, 25], a firewall analysis system that uses some specific firewall queries was presented. In [21,25], a firewall query is described by a triple (a set of source addresses, a set of destination addresses, a set of services), where each service is a tuple (protocol type, destination port number). The semantics of such a query are "which IP addresses in the set of source addresses can send which services in the set of services to which IP addresses in the set of destination addresses?". We go beyond [21, 25] in the following two major aspects.

1. No algorithm for processing a firewall query over a sequence of rules was presented in [21] or [25]. Consequently, how fast and scalable that a firewall query can be processed remains unknown, while the efficiency of a firewall query processing algorithm is crucial in order to interact with a human user. In contrast, we present an efficient algorithm for processing a firewall query over a sequence of rules. Our firewall query algorithm takes less than 10 milliseconds to process a query over a firewall that has up to 10,000 rules.
2. The query language described in [21] and [25] is too specific: it is only applicable to IP packets and it only concerns the four fields of source address, destination address, protocol type and destination port number. This makes

the expressive power of the query language in [21, 25] limited. For example, even only considering IP packets, it cannot express a firewall query concerning source port numbers or application fields. In contrast, our Structured Firewall Query Language is capable of expressing firewall queries with arbitrary fields.

In [18], some ad-hoc "what if" questions that are similar to firewall queries were discussed. However, no algorithm was presented for processing the proposed "what if" questions.

In [9], expert systems were proposed to analyze firewall rules. Clearly, building an expert system just for analyzing a firewall is overwrought and impractical.

Detecting potential firewall configuration errors by conflict detection was discussed in [3, 8, 17, 22]. Similar to conflict detection, six types of so-called "anomalies" were defined in [1]. Examining each conflict or anomaly is helpful in reducing potential firewall configuration errors; however, the number of conflicts or anomalies in a firewall is typically large, and the manual checking of each conflict or anomaly is unreliable because the meaning of each rule depends on the current order of the rules in the firewall, which may be incorrect.

Some firewall design methods have been proposed in [4, 13, 16, 20]. These works aim at creating firewall rules, while we aim at analyzing firewall rules.

Firewall vulnerabilities are discussed and classified in [11, 19]. However, the focus of [11, 19] are the vulnerabilities of the packet filtering software and the supporting hardware part of a firewall, not the configuration of a firewall.

There are some tools currently available for network vulnerability testing, such as Satan [10, 12] and Nessus [23]. 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.

3 Structured Firewall Query Language

3.1 Firewalls

In this section, we present the actual syntax of the firewall query language and show how to use this language to describe firewall queries.

We first define a *packet* over the fields $F_1, \cdots, F_d$ as a d-tuple $(p_1, \cdots, p_d)$ where each p_i is 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 $[0, 2^{32})$. For the brevity of presentation, we assume that all packets are over the d fields $F_1, \cdots, F_d$, if not otherwise specified. We use Σ to denote the set of all packets. It follows that Σ is a finite set and $|\Sigma| = |D(F_1)| \times \cdots \times |D(F_n)|$.

Given a firewall f, each packet p in Σ is mapped by f to a decision, denoted $f.p$, in the set $\{accept, discard\}$. Two firewalls f and f' are equivalent, denoted

$f \equiv f'$, iff for any packet p in Σ, $f.p = f'.p$ holds. This equivalence relation is symmetric, self-reflective, and transitive.

A firewall consists of a sequence of rules. Each rule is of the following format:

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

where each S_i is a nonempty subset of $D(F_i)$, and the $\langle decision \rangle$ is either *accept* or *discard*. If $S_i = D(F_i)$, we can replace $(F_i \in S_i)$ by $(F_i \in all)$, or remove the conjunct $(F_i \in D(F_i))$ altogether. Some existing firewall products, such as Linux's ipchain, require that S_i be represented in a prefix format such as 192.168.0.0/16, where 16 means that the prefix is the first 16 bits of 192.168.0.0 in a binary format. In this paper, we choose to represent S_i as a nonempty set of nonnegative integers because of two reasons. First, any set of nonnegative integers can be automatically converted to a set of prefixes (see [15]). Second, set representations are more convenient in mathematical manipulations.

A packet $(p_1, \cdots, p_d)$ *matches* a rule $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle decision \rangle$ iff the condition $(p_1 \in S_1) \wedge \cdots \wedge (p_d \in S_d)$ holds. Since a packet may match more than one rule in a firewall, each packet is mapped to the decision of the first rule that the packet matches. 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.

Here we give an example of a simple firewall. In this example, we assume that each packet only has two fields: S (source address) and D (destination address), and both fields have the same domain $[1, 10]$. This firewall consists of the sequence of rules in Figure 1. Let f_1 be the name of this firewall.

$r_1:\ S \in [4,7] \ \wedge\ D \in [6,8] \ \rightarrow\ accept$
$r_2:\ S \in [3,8] \ \wedge\ D \in [2,9] \ \rightarrow\ discard$
$r_3:\ S \in [1,10] \ \wedge\ D \in [1,10] \rightarrow\ accept$

Figure 1. Firewall f_1

3.2 Query Language

A *query*, denoted Q, in our Structured Firewall Query Language (SFQL) is of the following format:

select F_i
from f
where $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \wedge (\textbf{decision} = \langle dec \rangle)$

where F_i is one of the fields $F_1, \cdots, F_d$, f is a firewall, each S_j is a nonempty subset of the domain $D(F_j)$ of field F_j, and $\langle dec \rangle$ is either *accept* or *discard*.

The result of query Q, denoted $Q.result$, is the following set:

$\{p_i | (p_1, \cdots, p_d)$ *is a packet in* Σ, *and*
$(p_1 \in S_1) \wedge \cdots \wedge (p_d \in S_d) \wedge (f.(p_1, \cdots, p_d) = \langle dec \rangle)\}$

Recall that Σ denotes the set of all packets, and $f.(p_1, \cdots, p_d)$ denotes the decision to which firewall f maps the packet $(p_1, \cdots, p_d)$.

We can get the above set by first finding all the packets $(p_1, \cdots, p_d)$ in Σ such that the following condition

$$(p_1 \in S_1) \wedge \cdots \wedge (p_d \in S_d) \wedge (f((p_1, \cdots, p_d)) = \langle dec \rangle)$$

holds, then projecting all these packets to the field F_i.

For example, a question to the firewall in Figure 1, "Which computers whose addresses are in the set $[4, 8]$ can send packets to the machine whose address is 6?", can be formulated as the following query using SFQL:

select S
from f_1
where $(S \in \{[4, 8]\}) \wedge (D \in \{6\}) \wedge (\textbf{decision} = accept)$

The result of this query is $\{4, 5, 6, 7\}$.

As another example, a question to the firewall in Figure 1, "Which computer cannot send packets to the computer whose address is 6?", can be formulated as the following query using SFQL:

select S
from f_1
where $(S \in all) \wedge (D \in \{6\}) \wedge (\textbf{decision} = discard)$

The result of this query is $\{3, 8\}$.

Next we give more examples on how to use SFQL to describe firewall queries.

4 Firewall Query Examples

In this section, we describe some example firewall queries using SFQL. Let f be the name of the firewall that resides on the gateway router in Figure 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 these examples, we assume each packet has the following five fields: I (Interface), S (Source IP), D (Destination IP), N (Destination Port), P (Protocol Type).

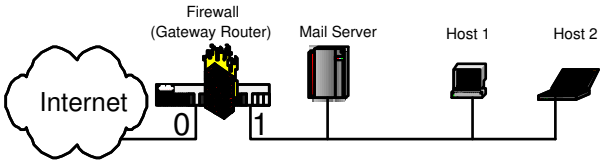

Figure 2. Firewall f

Question 1:
Which computers in the private network protected by the firewall f can receive BOOTP[2] packets from the outside Internet?
Query Q_1:
select D
from f
where $(I \in \{0\}) \wedge (S \in all) \wedge (D \in all) \wedge (N \in \{67, 68\})$
$\wedge (P \in \{udp\}) \wedge (\textbf{decision} = accept)$
Answer to question 1 is $Q_1.result$.

Question 2:
Which ports on the mail server protected by the firewall f are open?
Query Q_2:
select N
from f
where $(I \in \{0, 1\}) \wedge (S \in all) \wedge (D \in \{Mail\ Server\} \wedge (N \in all)$
$\wedge (P \in all) \wedge (\textbf{decision} = accept)$
Answer to question 2 is $Q_2.result$.

Question 3:
Which computers in the outside Internet cannot send SMTP[3] packets to the mail server protected by the firewall f?
Query Q_3:
select S
from f
where $(I \in \{0\}) \wedge (S \in all) \wedge (D \in \{Mail\ Server\}) \wedge (N \in \{25\})$
$\wedge (P \in \{tcp\}) \wedge (\textbf{decision} = discard)$
Answer to question 3 is $Q_3.result$.

[2] Bootp packets are UDP packets and use port number 67 or 68.

Question 4:
Which computers in the outside Internet cannot send any packet to the private network protected by the firewall f?
Query Q_4:
select S
from f
where $(I \in \{0\}) \wedge (S \in all) \wedge (D \in all) \wedge (N \in all) \wedge (P \in all)$
$\wedge(\textbf{decision} = accept)$
Answer to question 4 is $T - Q_4.result$, where T is the set of all IP addresses outside of the private network

Question 5:
Which computers in the outside Internet can send SMTP packets to both host 1 and host 2 in the private network protected by the firewall f?
Query Q_{5a}:
select S
from f
where $(I \in \{0\}) \wedge (S \in all) \wedge (D \in \{Host\ 1\}) \wedge (N \in \{25\})$
$\wedge(P \in \{tcp\}) \wedge (\textbf{decision} = accept)$
Query Q_{5b}:
select S
from f
where $(I \in \{0\}) \wedge (S \in all) \wedge (D \in \{Host\ 2\}) \wedge (N \in \{25\})$
$\wedge(P \in \{tcp\}) \wedge (\textbf{decision} = accept)$
Answer to question 5 is $Q_{5a}.result \cap Q_{5b}.result$.

5 Firewall Query Processing

In this section, we discuss how to process a firewall query for consistent firewalls. Consistent firewalls and inconsistent firewalls are defined as follows:

Definition 1 (Consistent Firewalls) A firewall is called a consistent firewall iff any two rules in the firewall do not conflict.

Definition 2 (Inconsistent Firewalls) A firewall is called an inconsistent firewall iff there are at least two rules in the firewall that conflict.

Recall that two rules in a firewall conflict iff they have different decisions and there is at least one packet that can match both rules. For example, the first two rules in the firewall in Figure 1, namely r_1 and r_2, conflict. Note that for any two rules in a consistent firewall, if they overlap, i.e., there is at least one packet can match both rules, they have the same decision. So, given a packet

[3] SMTP stands for Simple Mail Transfer Protocol. SMTP packets are TCP packets and use port number 25.

and a consistent firewall, all the rules in the firewall that the packet matches have the same decision. Figure 1 shows an example of an inconsistent firewall, and Figure 3 shows an example of a consistent firewall. In these two firewall examples, we assume that each packet only has two fields: S (source address) and D (destination address), and both fields have the same domain $[1, 10]$.

$$
\begin{array}{lllll}
r'_1: & S \in [4,7] & \wedge\ D \in [6,8] & \rightarrow a \\
r'_2: & S \in [4,7] & \wedge\ D \in [2,5] \cup [9,9] & \rightarrow d \\
r'_3: & S \in [4,7] & \wedge\ D \in [1,1] \cup [10,10] & \rightarrow a \\
r'_4: & S \in [3,3] \cup [8,8] & \wedge\ D \in [2,9] & \rightarrow d \\
r'_5: & S \in [3,3] \cup [8,8] & \wedge\ D \in [1,1] \cup [10,10] & \rightarrow a \\
r'_6: & S \in [1,2] \cup [9,10] & \wedge\ D \in [1,10] & \rightarrow a
\end{array}
$$

Figure 3. Consistent firewall f_2

Our interest in consistent firewalls is twofold. First, each inconsistent firewall can be converted to an equivalent consistent firewall, as described in Section 6. Second, as shown in the following theorem, it is easier to process queries for consistent firewalls than for inconsistent firewalls.

Theorem 1 (Firewall Query Theorem) Let Q be a query of the following form:

select F_i
from f
where $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \wedge (\textbf{decision} = \langle dec \rangle)$

If f is a consistent firewall that consists of n rules $r_1, \cdots, r_n$, then we have

$$Q.result = \bigcup_{j=1}^{n} Q.r_j$$

where each rule r_j is of the form

$$(F_1 \in S'_1) \wedge \cdots \wedge (F_d \in S'_d) \rightarrow \langle dec' \rangle$$

and the quantity of $Q.r_j$ is defined as follows:

$$Q.r_j = \begin{cases} S_i \cap S'_i & \text{if } (S_1 \cap S'_1 \neq \emptyset) \wedge \cdots \wedge (S_d \cap S'_d \neq \emptyset) \wedge (\langle dec \rangle = \langle dec' \rangle), \\ \emptyset & \text{otherwise} \end{cases}$$

□

The Firewall Query Theorem implies a simple query processing algorithm: given a consistent firewall f that consists of n rules $r_1, \cdots, r_n$ and a query Q, compute $Q.r_j$ for each j, then $\bigcup_{j=1}^{n} Q.r_j$ is the result of query Q. We call this algorithm the *rule-based firewall query processing algorithm*. Figure 4 shows the pseudocode of this algorithm.

Rule – based Firewall Query Processing Algorithm
Input : (1) A consistent firewall f that consists of n rules: $r_1, \cdots, r_n$,
(2) A query Q:
select F_i
from f
where $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \wedge (\textbf{decision} = \langle dec \rangle)$
Output: Result of query Q
Steps:
1. $Q.result := \emptyset$;
2. **for** $j := 1$ **to** n **do** /*Let $r_j = (F_1 \in S'_1) \wedge \cdots \wedge (F_d \in S'_d) \rightarrow \langle dec' \rangle$*/
if $(S_1 \cap S'_1 \neq \emptyset) \wedge \cdots \wedge (S_d \cap S'_d \neq \emptyset) \wedge (\langle dec \rangle = \langle dec' \rangle)$
then $Q.result := Q.result \cup (S_i \cap S'_i)$;
3. **return** $Q.result$;

Figure 4. Rule-based Firewall Query Processing Algorithm

6 FDT-based Firewall Query Processing Algorithm

Observe that multiple rules in a consistent firewall may share the same prefix. For example, in the consistent firewall in Figure 3, the first three rules, namely r'_1, r'_2, r'_3, share the same prefix $S \in [4, 7]$. Thus, if we apply the above query processing algorithm in Figure 4 to answer a query, for instance, whose "where clause" contains the conjunct $S \in \{3\}$, over the firewall in Figure 3, then the algorithm will repeat three times the calculation of $\{3\} \cap [4, 7]$. Clearly, repeated calculations are not desirable for efficiency purposes.

In this section, we present a firewall query processing method that has no repeated calculations and can be applied to both consistent and inconsistent firewalls. This method consists of two steps. First, convert the firewall (whether consistent or inconsistent) to an equivalent firewall decision tree (short for FDT). Second, use this FDT as the core data structure for processing queries. We call the algorithm that uses an FDT to process queries the *FDT-based firewall query processing algorithm*. Firewall decision trees are defined as follows. Note that firewall decision trees are a special type of firewall decision diagrams, which are introduced in [13] as a useful notation for specifying firewalls.

Definition 3 (Firewall Decision Tree) A Firewall Decision Tree t over fields $F_1, \cdots, F_d$ is a directed tree that has the following four properties:

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

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

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 5 shows an example of an FDT named t_3. In this example, we assume that each packet only has two fields: S (source address) and D (destination address), and both fields have the same domain $[1, 10]$. In the rest of this paper, including this example, we use "a" as a shorthand for *accept* and "d" as a shorthand for *discard*.

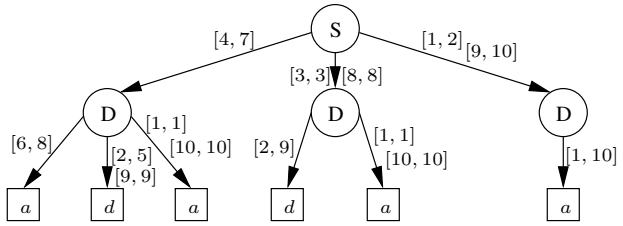

Figure 5. Firewall Decision Tree t_3

A decision path in an FDT t 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 FDT defines the following rule:

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

where

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

For an FDT t, we use $\Gamma(t)$ to denote the set of all the rules defined by all the decision paths of t. For any packet p, there is one and only one rule in Γ_t that p matches because of the consistency and completeness properties; therefore, t maps p to the decision of the only rule that p matches in Γ_t. Considering the FDT t_3 in Figure 5, Figure 3 shows all the six rules in Γ_{t_3}.

Given an FDT t, any sequence of rules that consists of all the rules in Γ_t is equivalent to t. The order of the rules in such a firewall is immaterial because the rules in Γ_t are non-overlapping. Given a sequence of rules, an equivalent FDT can be constructed using the construction algorithm described in [20]. Therefore, an inconsistent firewall can be converted to an equivalent consistent firewall using the following two steps: first, construct an equivalent FDT from the original inconsistent firewall; second, generate one rule for each decision path of the FDT. Then any sequence that consists of all the rules defined by the decision paths of the FDT is the resulting equivalent consistent firewall.

The pseudocode of the FDT-based firewall query processing algorithm is shown in Figure 6. Here we use $e.t$ to denote the (target) node that the edge e points to, and we use $t.root$ to denote the root of FDT t.

FDT – based Firewall Query Processing Algorithm
Input : (1)An FDT t,
(2)A query Q: **select** F_i
from t
where $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \wedge (\textbf{decision} = \langle dec \rangle)$
Output : Result of query Q
Steps:
1. $Q.result := \emptyset$;
2. **CHECK**($t.root$, $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \wedge (\textbf{decision} = \langle dec \rangle$)
3. **return** $Q.result$;

CHECK(v, $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \wedge (\textbf{decision} = \langle dec \rangle)$)
1. **if** (v is a terminal node) and ($F(v) = \langle dec \rangle$) **then**
   (1) Let $(F_1 \in S'_1) \wedge \cdots \wedge (F_d \in S'_d) \rightarrow \langle dec' \rangle$ be the rule defined by the decision path containing node v;
   (2) $Q.result := Q.result \cup (S_i \cap S'_i)$;
2. **if** (v is a nonterminal node) **then** /*Let F_j be the label of v*/
   for each edge e in $E(v)$ **do**
   if $I(e) \cap S_j \neq \emptyset$ **then**
   CHECK($e.t$, $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \wedge (\textbf{decision} = \langle dec \rangle)$)

Figure 6. FDT-based Firewall Query Processing Algorithm

The above FDT-based firewall query processing algorithm has two inputs, an FDT t and an SFQL query Q. The algorithm starts by traversing the FDT from its root. Let F_j be the label of the root. For each outgoing edge e of the root, we compute $I(e) \cap S_j$. If $I(e) \cap S_j = \emptyset$, we skip edge e and do not traverse the subgraph that e points to. If $I(e) \cap S_j \neq \emptyset$, then we continue to traverse the subgraph that e points to in a similar fashion. Whenever a terminal node is encountered, we compare the label of the terminal node and $\langle dec \rangle$. If they are the same, assuming the rule defined by the decision path containing the terminal node is $(F_1 \in S'_1) \wedge \cdots \wedge (F_d \in S'_d) \rightarrow \langle dec' \rangle$, then we add $S_i \cap S'_i$ to $Q.result$.

7 Experimental Results

So far we have presented two firewall query processing algorithms, the rule-based algorithm in Section 5 and the FDT-based algorithm in Section 6. In this section, we evaluate the efficiency of both algorithms. In the absence of publicly available firewalls, we create synthetic firewalls according to the characteristics of real-life packet classifiers discussed in [2, 14]. Note that a firewall is also a packet classifier. 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 of memory.

Figure 7 shows the average execution time of both algorithms versus the total number of rules in the original (maybe inconsistent) firewalls. The horizontal axis indicates the total number of rules in the original firewalls, and the vertical axis indicates the average execution time (in milliseconds) for processing a firewall query. Note that in Figure 7, the execution time of the FDT-based firewall query processing algorithm does not include the FDT construction time because the conversion from a firewall to an equivalent FDT is performed only once for each firewall, not for each query. Similarly, the execution time of the rule-based firewall query processing algorithm does not include the time for converting an inconsistent firewall to an equivalent consistent firewall because this conversion is performed only once for each firewall, not for each query.

From Figure 7, we can see that the FDT-based firewall query processing algorithm is much more efficient than the rule-based firewall query processing algorithm. For example, for processing a query over an inconsistent firewall that has 10,000 rules, the FDT-based query processing algorithm uses about 10 milliseconds, while the rule-based query processing algorithm uses about 100 milliseconds. The experimental results in Figure 7 confirm our analysis that the FDT-based query processing algorithm saves execution time by reducing repeated calculations.

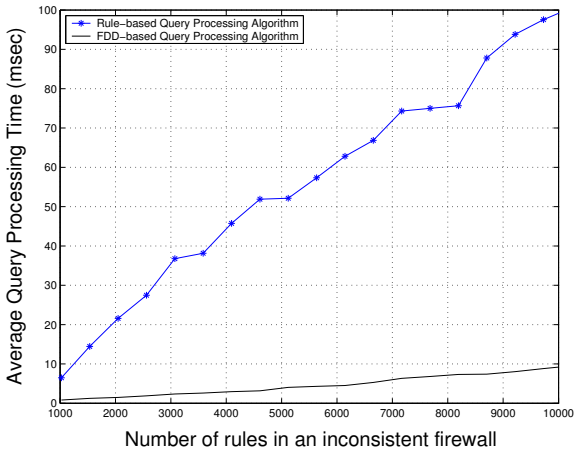

Figure 7. Query Processing Time vs. Number of rules

8 Concluding Remarks

Our contributions in this paper are three-fold. First, we introduce a simple and effective SQL-like query language, the Structured Firewall Query Language, for describing firewall queries. Second, we present a theorem, the Firewall Query Theorem, as the foundation for developing firewall query processing algorithms. Third, we present an efficient algorithm that uses firewall decision trees as its core data structure for processing firewall queries. Given a firewall of a sequence of rules, we first construct an equivalent firewall decision tree. Then the firewall decision tree is used as the core data structure of this query processing algorithm to answer each firewall query. Our experimental results show that this query processing algorithm is very efficient.

To keep our presentation simple, we have described a somewhat watered-down version of the firewall query language where the "select" clause in a query has only one field. In fact, the "select" clause in a query can be extended to have more than one field. The results in this paper, e.g., the Firewall Query Theorem and the two firewall query processing algorithms, can all be extended accordingly to accommodate the extended "select" clauses.

References

1. E. Al-Shaer and H. Hamed. Discovery of policy anomalies in distributed firewalls. In *IEEE INFOCOM'04*, March 2004.
2. F. Baboescu, S. Singh, and G. Varghese. Packet classification for core routers: Is there an alternative to cams? In *Proc. of IEEE INFOCOM*, 2003.

3. 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.
4. 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.
5. CERT. Test the firewall system. http://www.cert.org/security-improvement/practices/p060.html.
6. CERT Coordination Center. http://www.cert.org/advisories/ca-2003-20.html.
7. D. Moore et al. http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html.
8. D. Eppstein and S. Muthukrishnan. Internet packet filter management and rectangle geometry. In *Symp. on Discrete Algorithms*, pages 827–835, 2001.
9. 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.
10. 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.
11. 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.
12. M. Freiss. *Protecting Networks with SATAN*. O'Reilly & Associates, Inc., 1998.
13. M. G. Gouda and A. X. Liu. Firewall design: consistency, completeness and compactness. In *Proc. of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS'04)*, pages 320–327.
14. P. Gupta. *Algorithms for Routing Lookups and Packet Classification*. PhD thesis, Stanford University, 2000.
15. P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, 15(2):24–32, 2001.
16. J. D. Guttman. Filtering postures: Local enforcement for global policies. In *Proc. of IEEE Symp. on Security and Privacy*, pages 120–129, 1997.
17. A. Hari, S. Suri, and G. M. Parulkar. Detecting and resolving packet filter conflicts. In *Proc. of IEEE INFOCOM*, pages 1203–1212, 2000.
18. 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.
19. 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.
20. A. X. Liu and M. G. Gouda. Diverse firewall design. In *Proc. of the International Conference on Dependable Systems and Networks (DSN'04)*, pages 595–604, June 2004.
21. A. Mayer, A. Wool, and E. Ziskind. Fang: A firewall analysis engine. In *Proc. of IEEE Symp. on Security and Privacy*, pages 177–187, 2000.
22. J. D. Moffett and M. S. Sloman. Policy conflict analysis in distributed system management. *Journal of Organizational Computing*, 4(1):1–22, 1994.
23. Nessus. http://www.nessus.org/. March 2004.
24. A. D. Rubin, D. Geer, and M. J. Ranum. *Web Security Sourcebook*. Wiley Computer Publishing, 1th edition, 1997.
25. A. Wool. Architecting the lumeta firewall analyzer. In *Proc. of the 10th USENIX Security Symposium*, pages 85–97, August 2001.
26. A. Wool. A quantitative study of firewall configuration errors. *IEEE Computer*, 37(6):62–67, 2004.