Complete Redundancy Removal for Packet Classifiers in TCAMs

Alex X. Liu, *Member*, *IEEE*, and Mohamed G. Gouda, *Member*, *IEEE*

Abstract—Packet classification is the core mechanism that enables many networking services on the Internet such as firewall packet filtering and traffic accounting. Using Ternary Content Addressable Memories (TCAMs) to perform high-speed packet classification has become the de facto standard in the industry. TCAMs classify packets in constant time by comparing a packet with all classification rules of ternary encoding in parallel. Despite their high speed, TCAMs suffer from the well-known interval expansion problem. As packet classification rules usually have fields specified as intervals, converting such rules to TCAM-compatible rules may result in an explosive increase in the number of rules. This is not a problem if TCAMs have large capacities. Unfortunately, TCAMs have very limited capacity, and more rules means more power consumption and more heat generation for TCAMs. Even worse, the number of rules in packet classifiers have been increasing rapidly with the growing number of services deployed on the Internet. In this paper, we propose to address the interval expansion problem of TCAMs by removing redundant rules in classifiers. This equivalent transformation can significantly reduce the number of TCAM entries needed by a classifier. Our experiments on real-life classifiers show an average reduction of 58.2 percent in the number of TCAM entries by removing redundant rules. Given the logical interleaving nature of packet filtering rules, identifying redundant rules in classifiers is by no means trivial, and to achieve the guarantee of no redundant rules in resulting classifiers is even more challenging. In this paper, for the first time, we give a necessary and sufficient condition for identifying all redundant rules in a classifier. Based on this condition, we categorize redundant rules into upward redundant rules and downward redundant rules. Second, we present two algorithms for detecting and removing the two types of redundant rules, respectively. Third, we formally prove that the resulting classifiers have no redundant rules after running the two algorithms. Last, we conduct extensive experiments on both real-life and synthetic classifiers. The experimental results show that our redundancy removal algorithms are both effective and efficient.

Index Terms—Packet classification, Ternary Content Addressable Memory (TCAM), redundant rules.

1 Introduction

Packet classification, which has been widely deployed on the Internet, is the core mechanism that enables routers to perform many networking services such as firewall packet filtering, virtual private networks (VPNs), network address translation (NAT), quality of service (QoS), load balancing, traffic accounting and monitoring, differentiated services (Diffserv), etc. As more services are deployed on the Internet, packet classification grows in demand and importance.

The function of a packet classification system is to map each packet to a decision (i.e., action) according to a sequence (i.e., ordered list) of rules, which is called a *classifier*. Each rule in a classifier has a predicate over some packet header fields and a decision to be performed upon the packets that match the predicate. To resolve possible conflicts among rules in a classifier, the decision for each packet is the decision of the first (i.e., highest priority) rule that the packet matches. Table 1 shows an example classifier of three rules. The format of these rules is based upon the format used in Access Control Lists on Cisco routers.

1.1 Motivation

There are two types of packet classification schemes: software based and hardware based. Many advanced software-based packet classification algorithms and techniques have been proposed in the past decade (see the survey paper [37]). Based on complexity bounds from computational geometry [33], for packet classification with n rules and $d > 3$ fields, the "best" software-based packet classification algorithms use either $O(n^d)$ space and $O(\log n)$ time or $O(n)$ space and $O(\log^{d-1} n)$ time. Many software-based solutions are either too slow (such as linear search) or too memory intensive (such as RFC [16]). Decision-tree-based packet classification algorithms (such as [17] and [39]) seem to achieve better time-space trade-offs. However, they may not work as well in the future as they have exploited statistical characteristics of packet classifiers to achieve the above time-space trade-offs, and it has been observed that these statistical characteristics are changing [22].

Due to the inherent limitations of software-based packet classification algorithms, more and more packet classification systems are hardware based; specifically, most packet classification systems now use Ternary Content Addressable Memories (TCAMs). A TCAM is a memory chip where each entry can store a packet classification rule that is encoded in ternary format. Given a packet, the TCAM hardware can compare the packet with all stored rules in parallel and then return the decision of the first rule that the packet matches.

- *A.X. Liu is with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824. E-mail: alexliu@cse.msu.edu.*
- *M.G. Gouda is with the Department of Computer Sciences, University of Texas at Austin, 1 University Station (C0500), Austin, TX 78712-0233. E-mail: gouda@cs.utexas.edu.*

Manuscript received 12 Mar. 2008; revised 9 Sept. 2008; accepted 18 Sept. 2008; published online 24 Sept. 2008.
Recommended for acceptance by J.C.S. Lui.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-03-0097.
Digital Object Identifier no. 10.1109/TPDS.2008.216.

TABLE 1
An Example Classifier

Rule	Src IP	Dst IP	Src Port	Dst Port	Protocol	Action
r_1	*	192.168.0.1	*	*	*	discard
r_2	1.2.3.0/24	192.168.0.1	[1,65534]	[1,65534]	TCP	accept
r_3	*	*	*	*	*	accept

Thus, it takes $O(1)$ time to find the decision for any given packet. Current TCAMs can support up to 133 million searches per second for 144-bit-wide keys [22]. Because of their high speed, TCAMs have become the de facto industrial standard for high-speed packet classification [1], [3], [4], [22]. In 2003, most packet classification devices shipped were TCAM based [2]. More than six million TCAM devices were deployed worldwide in 2004 [2].

Despite their high speed, TCAMs have their own limitations with respect to packet classification:

1. *Interval expansion.* TCAMs can only store rules that are encoded in ternary format. In a typical packet classification rule, the source IP address, the destination IP address, and the protocol type are specified in prefix format, which can be directly stored in TCAMs, but source and destination port numbers are specified in intervals (i.e., ranges), which need to be converted to one or more prefixes before being stored in TCAMs. This can lead to a significant increase in the number of TCAM entries needed to encode a rule. For example, 30 prefixes are needed to represent the single interval [1, 65,534], so $30 \times 30 = 900$ TCAM entries are required to represent the single rule r_2 in Table 1.
2. *Low capacity.* TCAMs have limited capacity. The largest TCAM chip available on the market has 18 Mbits, while 2-Mbit and 1-Mbit chips are most popular [2]. Given that each TCAM entry has 144 bits and a packet classification rule may have a worst expansion factor of 900, it is possible that an 18-Mbit TCAM chip cannot store all the required entries for a modest classifier of only 139 rules. While the worst case may not happen in reality, this is certainly an alarming issue. Furthermore, TCAM capacity is not expected to increase dramatically in the near future due to other limitations that we will discuss next.
3. *High power consumption and heat generation.* TCAM chips consume large amounts of power and generate large amounts of heat due to their high circuit density. For example, a 1-Mbit TCAM chip consumes 15-30 W of power. Power consumption together with the consequent heat generation is a serious problem for core routers and other networking devices.
4. *Large board space occupation.* TCAMs occupy much more board space than SRAMs. For networking devices such as routers, area efficiency of the circuit board is a critical issue.
5. *High hardware cost.* TCAMs are expensive. For example, a 1-Mbit TCAM chip costs about 200 $\sim$ 250 US dollars. TCAM cost is a significant fraction of router cost.

All these limitations of TCAMs can be addressed by reducing the number of TCAM entries that a classifier requires. As we reduce the number of TCAM entries required, we can use TCAMs of smaller capacities, which results in less board space and lower hardware cost. Furthermore, reducing the number of rules in a TCAM directly reduces power consumption and heat generation because the energy consumed by a TCAM grows linearly with the number of ternary rules it stores [40].

$$
\begin{array}{l}
r_1 : \ F_1 \in [1, \ \ 50] \ \rightarrow accept \\
r_2 : \ F_1 \in [40, \ 90] \ \rightarrow discard \\
r_3 : \ F_1 \in [30, \ 60] \ \rightarrow discard \\
r_4 : \ F_1 \in [65, \ 95] \ \rightarrow accept \\
r_5 : \ F_1 \in [80, \ 100] \rightarrow accept
\end{array}
$$

Fig. 1. A simple classifier.

In this paper, we propose to reduce the number of TCAM entries that a classifier requires by removing the redundant rules in the classifier. A rule in a classifier is *redundant* if and only if removing the rule does not change the semantics of the classifier. For example, rule r_2 in the classifier in Table 1 is redundant because there is no packet whose first matching rule is r_2. Through this equivalent transformation of removing redundant rules, the number of TCAM entries needed by a classifier can be significantly reduced. Using the example of the classifier in Table 1, removing redundant rules reduces the number of TCAM entries needed from 902 to 2. Our experiments on real-life classifiers show an average reduction of 58.2 percent on the number of TCAM entries by removing redundant rules.

1.2 Redundant Rules

Classifiers often have redundant rules. A rule in a classifier is redundant if and only if removing the rule does not change the function of the classifier, i.e., does not change the decision of the classifier for every packet. For example, consider the classifier in Fig. 1, whose geometric representation is in Fig. 2. This classifier consists of five rules r_1 through r_5. For simplicity, we assume that this classifier only checks one packet field F_1 whose domain is [1, 100].

We have the following two observations concerning the redundant rules in the classifier in Fig. 1:

1. Rule r_3 is redundant. This is because the first matching rule for all packets where $F_1 \in [30, 50]$ is r_1, and the first matching rule for all packets where $F_1 \in [51, 60]$ is r_2. Therefore, there are no packets whose first matching rule is r_3. We call r_3 an upward redundant rule. A rule r in a classifier is *upward*

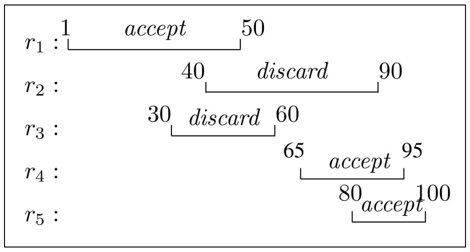


Fig. 2. Geometric representation of the classifier in Fig. 1.

redundant if and only if there are no packets whose first matching rule is r. Geometrically, a rule is upward redundant in a classifier if and only if the rule is overlayed by some rules listed above it.

2. Rule r_4 is redundant. This is because r_4 is the first matching rule for all packets where $F_1 \in [91, 95]$. If we remove rule r_4, the first matching rule for all those packets becomes r_5, which has the same decision as r_4. Therefore, removing rule r_4 does not change the function of the classifier. We call r_4 a downward redundant rule. A rule r in a classifier is *downward redundant* if and only if for each packet, whose first matching rule is r, the first matching rule below r has the same decision as r.

1.3 Redundancy Removal

A rule that examines d-dimensional fields can be viewed as a d-dimensional object. Real-life classifiers are typically four-dimensional or five-dimensional. While identifying redundant rules in one-dimensional classifiers (such as the one in Fig. 1) is simple, identifying redundant rules in multidimensional classifiers is by no means easy. In this paper, we first give a necessary and sufficient condition for identifying redundant rules, based on which we categorize redundant rules into upward redundant rules and downward redundant rules. We develop two algorithms for detecting and removing the two types of redundant rules, respectively. Our redundancy removal algorithms use a tree representation of classifiers, which is called firewall decision diagrams (FDDs). The upward redundancy removal algorithm scans the classifier top-down from the first rule to the last rule and, during the scanning, converts the classifier to an FDD and removes upward redundant rules. The downward redundancy removal algorithm scans the classifier bottom-up from the last rule to the first rule and, during the scanning, removes downward redundant rules. After the two-time scanning of the rules, the result classifier is guaranteed to be redundancy free.

1.4 Key Contributions

This paper represents the first technical study of redundant rules in classifiers. To summarize, we make the following key contributions:

1. We propose redundancy removal as a new approach to the interval expansion problem of TCAMs. Our experiments on real-life classifiers showed an average reduction of 58.2 percent on the number of TCAM entries by removing redundant rules. A key advantage of this approach is that it can be easily deployed because it does not require any modification of existing packet classification systems. In comparison, a number of previous interval expansion solutions require hardware and architecture modifications to existing packet classification systems, making their adoption by networking manufacturers and ISPs much less likely [22], [31], [35], [38].
2. We give a necessary and sufficient condition for identifying redundant rules for the first time. This condition lays the foundation for developing redundancy removal algorithms.
3. We present two tree-based algorithms for removing the two types of redundant rules that we define in this paper, respectively. We formally prove that the resulting classifiers have no redundant rules after running the two algorithms. The experiments that we conducted on both real-life and synthetic classifiers showed that removing all redundant rules from a large classifier with thousands of rules only takes a few seconds using the two algorithms.

Redundancy detection/removal has benefits beyond minimizing TCAM entries. One exemplary use of redundancy detection is in analyzing classifiers for potential errors. For instance, when a rule is shadowed by rules above it, the rule becomes redundant; however, this is typically not the intent of the router or firewall administrator. Therefore, redundancy could be an indicator of errors in classifiers.

The rest of this paper proceeds as follows: We start by reviewing previous work in Section 2. Then, we formally define the terms and concepts related to redundancy removal in Section 3. We give a necessary and sufficient condition for identifying redundant rules in Section 4. In Section 5, we introduce FDDs, which will be used as the core data structure for redundancy removal algorithms. The upward and downward redundancy removal algorithms are presented in Section 6 and 7, respectively. In Section 9, we show the experimental results on both real-life and synthetic classifiers. In Section 10, we discuss an open problem on redundancy removal. Finally, we give concluding remarks in Section 11.

2 Related Work

Many software solutions have been proposed for finding the decision of the first rule that a packet matches in a given classifier (see the survey paper [37]). A comprehensive survey of this work is given in [37].

Recently, hardware packet classification systems based on TCAMs have been widely deployed due to their $O(1)$ classification time. This has led to a significant amount of work that explores ways to cope with the well-known interval expansion problem. These solutions fall into three broad categories: 1) *TCAM modification*, which requires changing TCAM hardware circuits, 2) *range encoding*, which does not require changing TCAM hardware circuits but does require preprocessing for every packet, and 3) *classifier minimization*, which does not require changing TCAM hardware circuits nor preprocessing for any packet. Next, we review previous work in these three categories.

TCAM modification. The basic idea is to modify TCAM circuits for packet classification purposes. For example, Spitznagel et al. proposed adding comparators at each entry level to better accommodate range matching [35]. This is an important research direction. However, solutions from this research line are difficult to deploy due to issues of cost and development [22]. Furthermore, changing the ternary nature of TCAMs makes such TCAMs less generally applicable to applications other than packet classification. Additionally, there has been work on developing load balancing algorithms for TCAM-based systems by Zheng et al. [41], [42]. This work focuses on exploiting chip-level parallelism to increase classifier throughput with multiple TCAM chips without having to copy the complete classifier to every TCAM chip.

Range encoding. The basic idea is to reencode intervals that appear in a classifier and then store the reencoded rules in the TCAM. When a packet comes, the packet needs to be preprocessed according to the reencoding scheme such that the packet, after preprocessing, can be used as a search key for the TCAM. Previous range encoding schemes fall into two categories: database-independent encoding schemes [6], [22], where the encoding of each rule is independent of other rules in the classifier, and database-dependent encoding schemes [7], [31], [34], [38], where the encoding of each rule may depend on other rules in the classifier. While the TCAM circuit does not need to be modified to implement range encoding, the system hardware does need to be reconfigured to allow for preprocessing of packets, and the delay caused by packet preprocessing could be problematic.

Classifier minimization. The basic idea is to convert a given classifier to another semantically equivalent classifier that requires fewer TCAM entries. These solutions are the most likely to be deployed by networking vendors and ISPs because they require no changes to TCAM hardware or existing packet classification systems and incur no preprocessing overhead for packets. Our work, along with [5], [8], [9], and [36], falls into this category.

Three papers focus on one or two-dimensional classifiers. Draves et al. proposed an optimal solution for one-dimensional classifiers in the context of minimizing routing tables in [9]. Subsequently, in the same context of minimizing routing tables, Suri et al. proposed an optimal dynamic programming solution for one-dimensional classifiers. They also observed that a generalization of the dynamic program was optimal for two-dimensional classifiers in which either two rules are nonoverlapping or one contains the other geometrically [36]. Suri et al. noted that their dynamic program would not be optimal for classifiers with more than two dimensions. In our studies, we have extended and implemented Suri et al.'s algorithm to minimize five-dimensional classifiers. Unfortunately, the extended algorithm is prohibitively slow even for a classifier with just a few rules. Recently, Applegate et al. have proposed an optimal solution for classifiers with two dimensions where each rule must have one field specified as the whole domain of the field and there are only two decisions [5].

Our work is among the first in minimizing multi-dimensional classifiers in TCAMs with more than two dimensions. In [8], Dong et al. proposed schemes to reduce range expansion by repeatedly expanding or trimming ranges to prefix boundaries and then using our redundancy removal algorithms presented in [26], which is the preliminary version of this paper, as the core routine for testing whether a specific modification changes the semantics of a classifier. In [32], we proposed TCAM Razor, a greedy algorithm that finds locally minimal prefix solutions along each field and combines these solutions into a smaller equivalent prefix classifier. TCAM Razor uses our redundancy removal algorithms in [26] as an important post-processing procedure in minimizing classifiers. Although TCAM Razor achieves higher compression ratio than using redundancy removal alone, our redundancy removal algorithm can handle classifier updates more efficiently because redundancy removal does not rewrite any rule. In [30], Liu et al. presented an algorithm for compressing firewall rules. Although the compression algorithm in [30] can be used to compress general packet classifiers, it compresses rules specified in ranges, not in prefixes.

As a special type of classifiers, firewalls have been studied in previous work. Firewall policy design issues have been studied in [12], [13], [14], [25], and [27]. The analysis and verification methods of firewall policies have been presented in [11], [23], [24], and [28], and the testing of firewall policies was studied in [19]. Firewall vulnerabilities were discussed and classified in [10] and [21] with focus on firewall software. However, none of these works focuses on redundancy removal.

Our work is the first that can removal all redundant rules in a classifier. Little previous work is on redundancy removal. The only previous work that can be traced is Gupta's definitions of two special types of redundant rules in his thesis [15]: backward redundant rules and forward redundant rules. A rule r in a classifier is backward redundant if and only if there exists another rule r' listed above r such that all packets that match r also match r'. Clearly, a backward redundant rule is an upward redundant rule but not vice versa. For example, rule r_3 in Fig. 1 is upward redundant but not backward redundant. A rule r in a classifier is forward redundant if and only if there exists another rule r' listed below r such that the following three conditions hold: 1) all packets that match r also match r', 2) r and r' have the same decision, and 3) for each rule r'' listed between r and r', either r and r'' have the same decision or no packet matches both r and r''. Clearly, a forward redundant rule is a downward redundant rule but not vice versa. For example, rule r_4 in Fig. 1 is downward redundant but not forward redundant. *To summarize, backward redundant rules are a specifical type of upward redundant rules, and forward redundant rules are a specifical type of downward redundant rules.* No redundancy removal algorithms were given in [15], neither do the experimental results on TCAM reduction by removing redundant rules. Note that our work on redundancy removal has inspired some following work, such as [29], on this topic.

3 Formal Definitions

We now formally define the concepts of fields, packets, rules, classifiers, and redundant rules. A *field* F_i is a variable of finite length (i.e., of a finite number of bits). The domain of field F_i of w bits, denoted $D(F_i)$, is $[0, 2^w - 1]$. A *packet* over the d fields $F_1, \ldots, F_d$ is a d-tuple $(p_1, \ldots, p_d)$, where each p_i $(1 \le i \le d)$ is an element of $D(F_i)$. Classifiers usually check the following five fields: source IP address, destination IP address, source port number, destination port number, and protocol type. The lengths of these packet fields are 32, 32, 16, 16, and 8, respectively. We use Σ to denote the set of all packets over fields $F_1, \ldots, F_d$. It follows that Σ is a finite set, and $|\Sigma| = |D(F_1)| \times \cdots \times |D(F_d)|$, where $|\Sigma|$ denotes the number of elements in set Σ, and $|D(F_i)|$ denotes the number of elements in set $D(F_i)$.

A *rule* has the form $\langle predicate\rangle \rightarrow \langle decision\rangle$. A $\langle predicate\rangle$ defines a set of packets over the fields F_1 through F_d and is specified as $F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d$, where each S_i is a subset of $D(F_i)$ and is specified as either a prefix or a nonempty nonnegative integer interval. A *prefix* $\{0,1\}^k\{*\}^{w-k}$ with k leading 0s or 1s for a packet field of length w denotes the integer interval $[\{0,1\}^k\{0\}^{w-k}, \{0,1\}^k\{1\}^{w-k}]$. For example, prefix 01** denotes the interval [0100, 0111]. A rule $F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d \rightarrow \langle decision\rangle$ is a *prefix rule* if and only if each S_i is represented as a prefix.

When using a TCAM to implement a classifier, we typically require that all rules be prefix rules. However, in a typical classifier rule, some fields such as source and destination port numbers are represented as integer intervals rather than a prefix. This leads to *interval expansion* (also called *range expansion*), the process of converting a rule that may have fields represented as integer intervals into one or more prefix rules. In interval expansion, each field of a rule is first expanded separately. The goal is to find a minimum set of prefixes such that the union of the prefixes corresponds to the integer interval. For example, if one 3-bit field of a rule is the integer interval [1, 6], a corresponding minimum set of prefixes would be 001, 01 *, 10 *, 110. The worst case interval expansion of a w-bit integer interval results in a set containing $2w-2$ prefixes [18]. The next step is to compute the cross product of each set of prefixes for each field, resulting in a potentially large number of prefix rules. In Section 1, the interval expansion of rule r_2 in Table 1 resulted in $30 \times 30 = 900$ prefix rules.

A packet $(p_1, \ldots, p_d)$ *matches* a predicate $F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d$ and the corresponding rule if and only if the condition $p_1 \in S_1 \wedge \cdots \wedge p_d \in S_d$ holds. We use α to denote the set of possible values that $\langle decision\rangle$ can be. For firewalls, typical elements of α include accept, discard, accept with logging, and discard with logging.

A sequence of rules $\langle r_1, \ldots, r_n\rangle$ is *complete* if and only if for any packet p, there is at least one rule in the sequence that p matches. To ensure that a sequence of rules is complete and thus is a classifier, the predicate of the last rule is usually specified as $F_1 \in D(F_1) \wedge \cdots F_d \in \wedge D(F_d)$. A *classifier* f is a sequence of rules that is complete. A classifier f is a *prefix classifier* if and only if every rule in f is a prefix rule.

Two rules in a classifier may overlap; that is, there exists at least one packet that matches both rules. Furthermore, two rules in a classifier may conflict; that is, the two rules not only overlap but also have different decisions. Classifiers typically resolve conflicts by employing a first-match resolution strategy where the decision for a packet p is the decision of the first (i.e., highest priority) rule that p matches in f. The decision that classifier f makes for packet p is denoted $f(p)$.

We can think of a classifier f as defining a many-to-one mapping function from Σ to α, where Σ denotes the set of all possible packets, and α denotes the set of all possible decisions. Two classifiers f_1 and f_2 are *equivalent*, denoted $f_1 \equiv f_2$, if and only if they define the same mapping function from Σ to α; that is, for any packet $p \in \Sigma$, we have $f_1(p) = f_2(p)$. Using the concept of equivalent classifiers, we define redundant rules as follows:

Definition 1 (redundant rule). *A rule r is* redundant *in a classifier f if and only if the resulting classifier f' after removing rule r is equivalent to f.*

4 Redundancy of Classifiers

4.1 Matching Set and Resolving Set

Before introducing our redundancy theorem, we introduce two new concepts, matching set and resolving set, that are associated with each rule in a classifier.

Definition 2 (matching set and resolving set). *Consider a classifier f that consists of n rules $\langle r_1, r_2, \ldots, r_n\rangle$. The* matching set *of a rule r_i, denoted $M(r_i)$, is the set of all packets that match r_i. The* resolving set *of a rule r_i in this classifier, denoted $R(r_i, f)$, is the set of all packets that match r_i but do not match any r_j, where $1 \le j < i$.*

Note that the matching set of a rule depends only on the rule itself, while the resolving set of a rule depends on both the rule itself and all the rules listed above it in a classifier. For example, consider the rule r_2 in Fig. 1: its matching set is the set of all the packets whose F_1 field is in [40, 90]; its resolving set is the set of all the packets whose F_1 field is in [51, 90].

Based on Definition 2, the relationship between $M(r_i)$ and $R(r_i, f)$ is the following:

$$R(r_i, f) = M(r_i) - \bigcup_{j=1}^{i-1} M(r_j).$$

The following theorem, whose proof is given in Appendix A, states several important properties of matching sets and resolving sets.

Theorem 1 (resolving set theorem). *Let f be any classifier that consists of n rules: $\langle r_1, r_2, \ldots, r_n\rangle$. The following four conditions hold:*

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

The following corollary says that the last rule in a classifier can be modified in a way that the resulting classifier is equivalent to the original one.

Corollary 1. *Let f be any classifier that consists of n rules $\langle r_1, r_2, \ldots, r_n\rangle$. If rule r_n in f is of the form $(F_1 \in S_1) \wedge (F_2 \in S_2) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle decision\rangle$ and if f' is the resulting classifier after rule r_n is modified to become of the form $(F_1 \in D(F_1)) \wedge (F_2 \in D(F_2)) \wedge \cdots \wedge (F_d \in D(F_d)) \rightarrow \langle decision\rangle$, then f and f' are equivalent.*

By modifying rule r_n in this way, any postfix of a classifier is complete, i.e., if $\langle r_1, r_2, \ldots, r_n\rangle$ is complete, then $\langle r_i, r_{i+1}, \ldots, r_n\rangle$ is complete for each i, $1 \le i \le n$.

In the rest of this paper, we assume that the predicate of the last rule in a classifier is $(F_1 \in D(F_1)) \wedge (F_2 \in D(F_2)) \wedge \cdots \wedge (F_d \in D(F_d))$. This assumption is crucial in developing the redundancy theorem that we discuss next.

4.2 Redundancy Theorem

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

Theorem 2 (redundancy theorem). *Let f be any classifier that consists of n rules: $\langle r_1, r_2, \ldots, r_n\rangle$. A rule r_i is redundant in f if and only if one of the following two conditions holds:*

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

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

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

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

Consider the example classifier f in Fig. 1. Based on Corollary 1, we first modify the last rule r_5 to be $F_1 \in [1, 100] \rightarrow$ *accept* without changing the function of f. Let f' be the resulting classifier after the modification of the last rule. Looking upward, rule r_3 is upward redundant in f' because $R(r_3, f) = \emptyset$. Looking downward, rule r_4 is downward redundant in f' because $R(r_4, f')$ consists of all packets whose F_1 field is in [91, 95] and for any packet whose F_1 field is in [91, 95], $\langle r_5\rangle(p)$ is *accept*, which is the same as the decision of r_4.

5 Firewall Decision Diagrams and Rules

In [14], Gouda and Liu presented FDDs as a useful notation for specifying firewalls. In this paper, we extend these diagrams to specify classifiers; therefore, we call the extended decision diagrams FDDs. Later, we show that Packet Decisions Diagrams play an important role in our redundancy removal algorithms.

Definition 4. *An FDD f with a decision set DS and over fields $F_1, \ldots, F_d$ 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 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*

$$F(v) \in \begin{cases} \{F_1, \ldots, F_d\}, & \text{if } v \text{ is a nonterminal node,} \\ DS, & \text{if } v \text{ is a terminal node.} \end{cases}$$

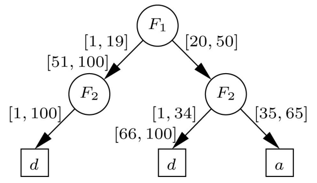


Fig. 3. An FDD.

3. *Each edge e in f has a label, denoted $I(e)$, such that if e is an outgoing edge of node v, then $I(e)$ is a nonempty subset of $D(F(v))$.*
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. *$\bigcup_{e \in E(v)} I(e) = D(F(v))$.*

Fig. 3 shows an example of an FDD with a decision set $\{a, d\}$ and over the two fields F_1 and F_2, where $D(F_1) = D(F_2) = [1, 100]$. In the examples in this paper, we employ the decision set $\{a, d\}$, where "a" represents "accept" and "d" represents "discard."

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

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

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 of the FDD f; therefore, f maps p to the decision of the only rule that p matches in S_f. We use $f(p)$ to denote the decision to which an FDD f maps a packet p. An FDD f and a sequence of rules f' are equivalent, denoted $f \equiv f'$, if and only if for any packet p, the condition $f(p) = f'(p)$ holds.

Given an FDD f, any classifier that consists of all the rules in S_f is equivalent to f. The order of the rules in such a classifier is immaterial because there are no overlapping rules in S_f.

Given a sequence of rules, in Section 6, we will see that an equivalent FDD is constructed after all the upward redundant rules are removed by the upward redundancy removal algorithm.

In the process of detecting and removing downward redundant rules, the data structure that we maintain is called a standard FDD. A *standard* FDD is a special type of FDD where the following two additional conditions hold:

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

Fig. 4. A classifier of four rules.

1. Each node has at most one incoming edge (i.e., a standard FDD is of a tree structure).
2. Each decision path contains d nonterminal nodes, and the ith node is labeled F_i for each i, where $1 \le i \le d$ (i.e., each decision path in a standard FDD is of the form $(v_1 e_1 v_2 e_2 \dots v_d e_d v_{d+1})$, where $F(v_i) = F_i$ for each i, where $1 \le i \le d$).

An example of a standard FDD is given in Fig. 3.

In the process of checking upward redundant rules, the data structure that we maintain is called a partial FDD. A *partial* FDD is a diagram that may not have the completeness property of a standard FDD but has all the other properties of a standard FDD.

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

Given a partial FDD f and a sequence of rules $\langle r_1, r_2, \dots, r_k \rangle$ that may not be complete, we say f is *equivalent* to $\langle r_1, r_2, \dots, r_k \rangle$ if and only if the following two conditions hold:

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

6 Removing Upward Redundancy

In this section, we discuss how to remove upward redundant rules. By definition, a rule is upward redundant if and only if its resolving set is empty. Therefore, in order to remove all upward redundant rules from a classifier, we need to calculate the resolving set for each rule in the classifier. The resolving set of each rule is calculated by its effective rule set. An effective rule set of a rule r in a classifier f is a set of rules where the union of all the matching sets of these rules is exactly the resolving set of rule r in f. More precisely, an effective rule set of a rule r is defined as follows:

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

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

For example, consider the classifier in Fig. 1. Then, $\{F_1 \in [1,50] \rightarrow accept\}$ is an effective rule set of rule r_1, $\{F_1 \in [51,90] \rightarrow discard\}$ is an effective rule set of rule r_2, $\emptyset$ is an effective rule set of rule r_3, and $\{F_1 \in [91,95] \rightarrow discard\}$ is an effective rule set of rule r_4. Clearly, once we obtain an effective rule set of a rule r in a classifier f, we know the resolving set of the rule r in f and consequently know whether the rule r is upward redundant in f. Note that by the definition of effective rule sets, if one effective rule set of a rule r is empty, then any effective rule set of the rule r is empty. Theorem 3 follows from the above discussion.

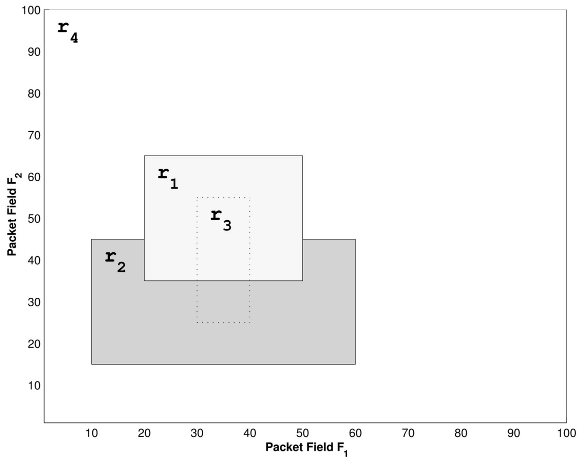


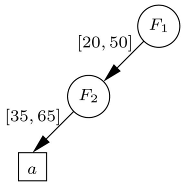
Fig. 5. Geometric representation of the rules in Fig. 4.

Theorem 3. *A rule r is upward redundant in a classifier if and only if an effective rule set of r is empty.*

Based on Theorem 3, the basic idea of our upward redundancy removal algorithm is given as follows: Given a classifier $\langle r_1, r_2, \dots, r_n \rangle$, we calculate an effective rule set for each rule from r_1 to r_n. If the effective rule set calculated for a rule r_i is empty, then r_i is upward redundant and is removed. Now, the problem is how to calculate an effective rule set for each rule in a classifier.

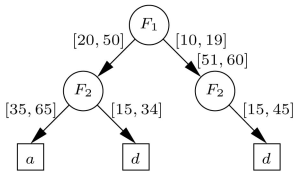
An effective rule set for each rule in a classifier is calculated with the help of partial FDDs. Consider a classifier that consists of n rules $\langle r_1, r_2, \dots, r_n \rangle$. Our upward redundancy removal algorithm first builds a partial FDD, denoted f_1, that is equivalent to the sequence $\langle r_1 \rangle$ and calculates an effective rule set, denoted E_1, of rule r_1. (Note that E_1 cannot be empty because $M(r_1) \neq \emptyset$; therefore, r_1 cannot be upward redundant.) Then, the algorithm transforms the partial FDD f_1 to another partial FDD, denoted f_2, that is equivalent to the sequence $\langle r_1, r_2 \rangle$ and, during the transformation process, calculates an effective rule set, denoted E_2, of rule r_2. The same transformation process continues until we reach r_n. When we finish, an effective rule set is calculated for each rule.

Here, we use f_i to denote the partial FDD that we constructed from the rule sequence $\langle r_1, r_2, \dots, r_i \rangle$ and E_i to denote the effective rule set that we calculated for rule r_i. By the following example, we show the process of transforming the partial FDD f_i to the partial FDD f_{i+1} and the calculation of E_{i+1}. Consider the classifier in Fig. 4 with the decision set $\{a, d\}$ and over fields F_1 and F_2, where $D(F_1) = D(F_2) = [1, 100]$. Fig. 5 shows the geometric representation of this classifier, where each rule is represented by a rectangle. In Fig. 5, we can see that rule r_3 is upward redundant because r_3, whose area is marked by dashed lines, is totally overlaid by rules r_1 and r_2. Later, we will see that the effective rule set calculated by our upward redundancy removal algorithm for rule r_3 is indeed an empty set.



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

Fig. 6. Partial FDD f_1 and an effective rule set E_1 of rule r_1 in Fig. 4.



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

Fig. 7. Partial FDD f_2 and an effective rule set E_2 of rule r_2 in Fig. 4.

Fig. 6 shows a partial FDD f_1 that is equivalent to $\langle r_1 \rangle$ and an effective rule set E_1 of rule r_1. In this figure, we use v_1 to denote the node with label F_1, e_1 to denote the edge with label [20, 50], and v_2 to denote the node with label F_2.

Now, we show how to append rule r_2 to f_1 in order to get a partial FDD f_2 that is equivalent to $\langle r_1, r_2 \rangle$ and how to calculate an effective rule set E_2 of rule r_2. We first compare the set [10, 60] with the set [20, 50] labeled on the outgoing edge of v_1. Since $[10, 60] - [20, 50] = [10, 19] \cup [51, 60]$, r_2 is the first matching rule for all packets that satisfy $F_1 \in [10, 19] \cup [51, 60] \wedge F_2 \in [15, 45]$, so we add one outgoing edge e to v_1, where e is labeled $[10, 19] \cup [51, 60]$, and e points to the path built from $F_2 \in [15, 45] \rightarrow d$. The rule defined by the decision path containing e, $F_1 \in [10, 19] \cup [51, 60] \wedge F_2 \in [15, 45] \rightarrow d$, should be put in E_2 because for all packets that match this rule, r_2 is their first matching rule. Because $[20, 50] \subset [10, 60]$, r_2 is possibly the first matching rule for a packet that satisfies $F_1 \in [20, 50]$. We further compare the set [35, 65] labeled on the outgoing edge of v_2 with the set [15, 45]. Since $[15, 45] - [35, 65] = [15, 34]$, we add a new edge e' to v_2, where e' is labeled [15, 34], and e' points to a terminal node labeled d. Similar to what we did to the new edge added to node v_1, we add the rule, $F_1 \in [20, 50] \wedge F_2 \in [15, 34] \rightarrow d$, defined by the decision path containing the new edge e' into E_2. The partial FDD f_2 and an effective rule set E_2 of rule r_2 are shown in Fig. 7, where E_2 consists of the two rules defined by the two new edges e and e' that we add to the partial FDD f_1 in Fig. 6.

Let f be any classifier that consists of n rules: $\langle r_1, r_2, \ldots, r_n \rangle$. A partial FDD that is equivalent to $\langle r_1 \rangle$ is easy to construct. Assume that r_1 is $(F_1 \in S_1) \wedge (F_2 \in S_2) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle decision \rangle$. Then, the partial FDD that consists of only one path $(v_1 e_1 v_2 e_2 \cdots v_d e_d v_{d+1})$, where $F(v_i) = F_i$ and $I(e_i) = S_i$ for $1 \leq i \leq d$ and $F(v_{d+1}) = \langle decision \rangle$, is equivalent to $\langle r_1 \rangle$. We denote this partial FDD by f_1 and call $(v_1 e_1 v_2 e_2 \cdots v_d e_d v_{d+1})$ *the path that is built from rule* $(F_1 \in S_1) \wedge (F_2 \in S_2) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle decision \rangle$.

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

First, we examine whether we need to add another outgoing edge to v. If $S_1 - (I(e_1) \cup I(e_2) \cup \cdots \cup I(e_k)) \neq \emptyset$, we need to add a new outgoing edge e_{k+1} with label $S_1 - (I(e_1) \cup I(e_2) \cup \cdots \cup I(e_k))$ to v. This is because any packet whose F_1 field satisfies $S_1 - (I(e_1) \cup I(e_2) \cup \cdots \cup I(e_k))$ does not match any of the first i rules but matches r_{i+1} provided that the packet also satisfies $(F_2 \in S_2) \wedge (F_3 \in S_3) \wedge \cdots \wedge (F_d \in S_d)$. The new edge e_{k+1} points to the root of the path that is built from $(F_2 \in S_2) \wedge (F_3 \in S_3) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle decision \rangle$. The rule r, $(F_1 \in S_1 - (I(e_1) \cup I(e_2) \cup \cdots \cup I(e_k))) \wedge (F_2 \in S_2) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle decision \rangle$, defined by the decision path containing the new edge e_{k+1} has the property $M(r) \subseteq R(r_{i+1}, f)$. Therefore, we add rule r to E_i.

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

1. $S_1 \cap I(e_j) = \emptyset$. In this case, we skip edge e_j because any packet whose value of field F_1 is in set $I(e_j)$ does not match r_{i+1}.
2. $S_1 \cap I(e_j) = I(e_j)$. In this case, for a packet p whose value of field F_1 is in set $I(e_j)$, the first rule that p matches may be one of the first i rules and may be rule r_{i+1}. Therefore, we append $(F_2 \in S_2) \wedge (F_3 \in S_3) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle decision \rangle$ to the subgraph rooted at the node that e_j points to in a similar fashion.
3. $S_1 \cap I(e_j) \neq \emptyset$ and $S_1 \cap I(e_j) \neq I(e_j)$. In this case, we split edge e into two edges: e' with label $I(e_j) - S_1$ and e'' with label $I(e_j) \cap S_1$. Then, we make two copies of the 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 process of appending rule r_{i+1} to partial FDD f_i, each time when we add a new edge to a node in f_i, the rule defined by the decision path containing the new edge is added to E_{i+1}. After the partial FDD f_i is transformed to f_{i+1}, the rules in E_{i+1} satisfy the following two conditions: 1) the union of all the matching sets of these rules is the resolving set of r_{i+1} according to the transformation process, and 2) all these rules have the same decision as r_{i+1} according to the transformation process. Therefore, E_{i+1} is an effective rule set of rule r_{i+1}.

By applying our upward redundancy removal algorithm to the classifier in Fig. 4, we get an effective rule set for each rule, as shown in Fig. 8. Note that $E_3 = \emptyset$, which means that rule r_3 is upward redundant; therefore, r_3 is removed.

The pseudocode for removing upward redundant rules is shown in Fig. 9. In the algorithm, we use $e.t$ to denote the node that edge e points to.

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

Fig. 8. Effective rule sets calculated for the classifier in Fig. 4.

```
Upward Redundancy Removal Algorithm
input : A classifier $f$ that consists of $n$ rules
        $\langle r_1, r_2 \cdots, r_n \rangle$
output: (1) Upward redundant rules in $f$ are removed.
        (2) An effective rules set for each rule is calculated.

1. Build a path from rule $r_1$ and let $v$ be the root;
   $E_1 := \{r_1\}$;
2. for $i := 2$ to $n$ do
     (1) $E_i := \emptyset$;
     (2) Ecal( $v$, $i$, $r_i$ );
     (3) if $E_i = \emptyset$ then remove $r_i$;

Ecal( $v$, $i$, $(F_j \in S_j) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle decision \rangle$ )
/*$F(v) = F_j$ and $E(v) = \{e_1, \cdots, e_k\}$*/
1. if $S_j - ( I(e_1) \cup \cdots \cup I(e_k) ) \neq \emptyset$ then
     (1) Add an outgoing edge $e_{k+1}$ with label
           $S_j - (I(e_1) \cup \cdots \cup I(e_k))$ to $v$;
     (2) Build a path from
           $(F_{j+1} \in S_{j+1}) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle decision \rangle$,
           and let $e_{k+1}$ point to its root;
     (3) Add the rule defined by the decision path
           containing edge $e_{k+1}$ to $E_i$;
2. if $j < d$ then
     for $g := 1$ to $k$ do
       if $I(e_g) \subseteq S_j$ then
         Ecal( $e_g.t$, $i$, $(F_{j+1} \in S_{j+1}) \wedge \cdots \wedge (F_d \in S_d)$
                                            $\rightarrow \langle decision \rangle$ );
       else if $I(e_j) \cap S_i \neq \emptyset$ then
           (1) $I(e_g) := I(e_g) - S_j$;
           (2) Add one outgoing edge $e$ with label
                 $I(e_g) \cap S_j$ to $v$;
           (3) Replicate the graph rooted at $e_g.t$, and
                 let $e$ points to the replicated graph;
           (4) Ecal( $e.t$, $i$, $(F_{j+1} \in S_{j+1}) \wedge \cdots$
                          $\wedge (F_d \in S_d) \rightarrow \langle decision \rangle$ );
```

Fig. 9. Upward redundancy removal algorithm.

7 Removing Downward Redundancy

The effective rule set E_i calculated for rule r_i in a classifier f is useful in checking whether r_i is downward redundant because the resolving set of r_i in f can be easily obtained by the union of the matching set of every rule in E_i.

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

Theorem 4. *Let f be any classifier that consists of n rules: $\langle r_1, r_2, \ldots, r_n \rangle$. Let f_i $(2 \leq i \leq n)$ be a standard FDD that is equivalent to the sequence of rules $\langle r_i, r_{i+1}, \ldots, r_n \rangle$. The rule r_{i-1} with an effective rule set E_{i-1} is downward redundant in f if and only if for each rule r in E_{i-1} and for each decision path $(v_1 e_1 v_2 e_2 \cdots v_d e_d v_{d+1})$ in f_i where rule r overlaps the rule that is defined by this decision path, the decision of r is the same as the label of the terminal node v_{d+1}.*

Proof sketch. Since the sequence of rules $\langle r_i, r_{i+1}, \ldots, r_n \rangle$ is complete, there exists a standard FDD that is equivalent to this sequence of rules. By the redundancy theorem, rule r_{i-1} is downward redundant if and only if for each rule r in E_{i-1} and for any p, where $p \in M(r)$, $\langle r_i, r_{i+1}, \ldots, r_n \rangle(p)$ is the same as the decision of r. Therefore, Theorem 4 follows. □

Now, we consider how to construct a standard FDD f_i, $2 \leq i \leq n$, that is equivalent to the sequence of rules $\langle r_i, r_{i+1}, \ldots, r_n \rangle$. The standard FDD f_n can be built from rule r_n in the same way that we build a path from a rule in the upward redundancy removal algorithm.

Suppose we have constructed a standard FDD f_i that is equivalent to the sequence of rules $\langle r_i, r_{i+1}, \ldots, r_n \rangle$. First, we check whether rule r_{i-1} is downward redundant by Theorem 4. If rule r_{i-1} is downward redundant, then we remove r_{i-1}, rename the standard FDD f_i to be f_{i-1}, and continue to check whether r_{i-2} is downward redundant. If rule r_{i-1} is not downward redundant, then we append rule r_{i-1} to the standard FDD f_i such that the resulting diagram is a standard FDD, denoted f_{i-1}, that is equivalent to the sequence of rules $\langle r_{i-1}, r_i, \ldots, r_n \rangle$. This procedure of transforming a standard FDD by appending a rule is similar to the procedure of transforming a partial FDD in the upward redundancy removal algorithm. The above process continues until we reach r_1; therefore, all downward rules are removed. The pseudocode for detecting and removing downward redundant rules is shown in Fig. 10.

Applying our downward redundancy removal algorithm to the classifier in Fig. 4, assuming that r_3 has been removed, rule r_2 is detected to be downward redundant; therefore, r_2 is removed. The standard FDD in Fig. 3 is the resulting standard FDD by appending rule r_1 to the standard FDD that is equivalent to $\langle r_4 \rangle$.

8 Complete Redundancy Removal Algorithm

Given a classifier, our redundancy removal algorithm consists of the following two steps (the resulting classifier after the two steps is guaranteed to be redundancy free):

1. Scan the classifier top-down to remove upward redundant rules using the upward redundancy removal algorithm.
2. Scan the classifier bottom-up to remove downward redundant rules using the downward redundancy removal algorithm.

Theorem 5. *Given a classifier f, let f' be the resulting classifier after applying the upward redundancy removal algorithm on f and f'' be the resulting classifier after applying the downward redundancy removal algorithm on f'. Then, there is no rule in f'' that is redundant.*

Proof sketch. Consider a rule r_i in f''. Let g denote the classifier after removing the downward redundant rules below r_i in f'. The difference between g and f'' is that the downward redundant rules above r_i are removed in f''. Thus, $R(r_i, g) \subseteq R(r_i, f'')$. Since $R(r_i, g) \neq \emptyset$, we have

```
Downward Redundancy Removal Algorithm
input  : A classifier $\langle r_1, r_2 \cdots, r_n\rangle$ where
         each rule $r_i$ has an effective rule set $E_i$.
output : Downward redundant rules in $f$ are removed.

1. Build a path from rule $r_n$ and let $v$ be the root;
2. for $i := n-1$ to 1 do
     if IsDownwardRedundant( $v$, $E_i$ ) = $true$
     then remove $r_i$;
     else Append( $v$, $r_i$ );

IsDownwardRedundant( $v$, $E$ ) /*$E = \{r'_1, \cdots, r'_m\}$*/
1. for $j := 1$ to $m$ do
     if HaveSameDecision( $v$, $r'_j$ ) = $false$ then
       return( $false$ );
2. return( $true$ );

HaveSameDecision( $v$, $(F_i \in S_i) \wedge \cdots \wedge (F_d \in S_d)$
                                   $\rightarrow \langle decision\rangle$ )
/*$F(v) = F_i$ and $E(v) = \{e_1, \cdots, e_k\}$*/
1. for $j := 1$ to $k$ do
     if $I(e_j) \cap S_i \neq \emptyset$ then
       if $i < d$ then
         if HaveSameDecision($e_j.t$, $(F_{i+1} \in S_{i+1})$
            $\wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle decision\rangle$ ) = false
         then return( $false$ );
       else
         if $F(e_j.t) \neq \langle decision\rangle$ then return( $false$ );
2. return( $true$ );

Append( $v$, $(F_i \in S_i) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle decision\rangle$ )
/*$F(v) = F_i$ and $E(v) = \{e_1, \cdots, e_k\}$*/
if $i < d$ then
  for $j := 1$ to $k$ do
    if $I(e_j) \subseteq S_i$ then
      Append( $e_j.t$, $(F_{i+1} \in S_{i+1}) \wedge \cdots \wedge (F_d \in S_d)$
              $\rightarrow \langle decision\rangle$ );
    else if $I(e_j) \cap S_i \neq \emptyset$ then
            (1) $I(e_j) := I(e_j) - S_i$;
            (2) Add one outgoing edge $e$ with label
                $I(e_j) \cap S_i$ to $v$;
            (3) Replicate the graph rooted at $e_j.t$, and
                let $e$ points to the replicated graph;
            (4) Append( $e.t$, $(F_{i+1} \in S_{i+1}) \wedge \cdots$
                        $\wedge (F_d \in S_d) \rightarrow \langle decision\rangle$ );
else /*$i = d$*/
  (1) for $j := 1$ to $k$ do
        (a) $I(e_j) := I(e_j) - S_i$;
        (b) if $I(e_j) = \emptyset$ then remove edge $e_i$ and node $e_i.t$;
  (2) Add one outgoing edge $e$ with label $S_i$ to $v$,
      create a terminal node with label $\langle decision\rangle$,
      and let $e$ point this terminal node;
```

Fig. 10. Downward redundancy removal algorithm.

$R(r_i, f'') \neq \emptyset$. Thus, r_i is not upward redundant in f''. Because r_i is not downward redundant in g, there exists at least one packet p in $R(r_i, g)$ that will be resolved differently by the rules below r_i in g if we remove r_i from g. Because $p \in R(r_i, g) \subseteq R(r_i, f'')$ and the rules below r_i are the same in both g and f'', p will be resolved differently by the rules below r_i in f'' if we remove r_i from f''. Thus, r_i is not downward redundant in f''. Therefore, Theorem 5 follows. □

Let n be the total number of rules in a classifier and d be the total number of distinct packet fields that are examined by a classifier. The time and space complexity of the redundancy removal algorithm is $O(n^d)$. Despite the high worst case complexities, our algorithms are practical for two reasons. First, d is typically small. Real-life classifiers typically examine five packet fields: source IP address, destination IP address, source port number, destination port number, and protocol type. Second, the worst cases of our algorithms are extremely unlikely to happen in practice. The experimental results have confirmed the above observations.

The time and space complexities of the two redundancy removal algorithms lie in the time and space complexities of the FDD construction algorithm [27], which is $O(n^d)$ in the worst case, where n is the number of rules and d is the number of fields. Note that d is typically at most 5 (source IP, source port, destination IP, destination port, protocol type).

9 Experimental Results

In this section, we evaluate the effectiveness and efficiency of the redundancy removal algorithm on both real-life and synthetic classifiers.

9.1 Metrics

We first define the metrics that we used to measure the effectiveness of the redundancy removal algorithm. In this paragraph, f denotes a classifier, S denotes a set of classifiers, and RR denotes the redundancy removal algorithm. We let $RR(f)$ denote the classifier produced by applying the redundancy removal algorithm on f, $Direct(f)$ denote the prefix classifier produced by applying direct interval expansion on f, and $|f|$ denote the number of rules in f. We define the following three metrics for assessing the performance of RR on a classifier f:

- The *redundancy ratio* of RR over $f = \frac{|f|-|RR(f)|}{|f|}$.
- The *compression ratio* of RR over $f = \frac{|Direct(RR(f))|}{|Direct(f)|}$.
- The *expansion ratio* of RR over $f = \frac{|Direct(RR(f))|}{|f|}$.

We define the following four metrics for assessing the performance of RR on a set of classifiers S:

- The $average\ redundancy\ ratio = \frac{\Sigma_{f\in S}\frac{|f|-|RR(f)|}{|f|}}{|S|}$.
- The $total\ redundancy\ ratio = \frac{\Sigma_{f\in S}|f|-|RR(f)|}{\Sigma_{f\in S}|f|}$.
- The $average\ compression\ ratio = \frac{\Sigma_{f\in S}\frac{|Direct(RR(f))|}{|Direct(f)|}}{|S|}$.
- The $total\ compression\ ratio = \frac{\Sigma_{f\in S}|Direct(RR(f))|}{\Sigma_{f\in S}|Direct(f)|}$.

For comparison purposes, we measure the following two metrics for direct expansion on a set of classifiers S:

- The *average expansion ratio* of *direct expansion* over $S = \frac{\Sigma_{f\in S}\frac{|Direct(f)|}{|f|}}{|S|}$.
- The *total expansion ratio* of *direct expansion* over $S = \frac{\Sigma_{f\in S}|Direct(f)|}{\Sigma_{f\in S}|f|}$.

In our experiments, we treat redundancy removal as a classifier compression scheme. Note that our redundancy removal algorithm can be used as a preprocessing or postprocessing procedure for other classifier compression schemes. For example, the TCAM Razor algorithm in [32] uses redundancy removal as a critical postprocessing procedure.

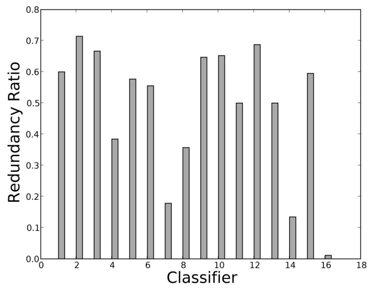


Fig. 11. Redundancy ratios of real-life classifiers.

9.2 Effectiveness on Real-Life Classifiers

We use RL to denote the set of 17 real-life classifiers that we performed experiments on. We actually obtained 42 real-life classifiers from distinct network service providers that range in size from dozens to hundreds of rules. Although this collection of classifiers is diverse, some classifiers from the same network service provider have similar structures and exhibited similar results under the redundancy removal algorithm. To prevent this repetition from skewing the performance data, we divided the 42 classifiers into 17 structurally distinct groups, and we randomly chose one from each of the 17 groups to form the set RL.

Redundancy ratio of real-life classifiers. Fig. 11 shows the redundancy ratios of the redundancy removal algorithm for each classifier in RL. The average and total redundancy ratios of the classifiers in RL are 45.7 percent and 24.0 percent, respectively. This shows that a significant percentage of the rules in real-life classifiers are redundant.

Compression ratio of real-life classifiers. Fig. 12 shows the compression ratios of the redundancy removal algorithm for each classifier in RL. The average and total compression ratios of the redundancy removal algorithm over the classifiers in RL are 41.8 percent and 35 percent, respectively. We can see that our redundancy removal algorithm can significantly reduce the number of TCAM entries for a classifier.

Expansion ratio of real-life classifiers. Fig. 13 shows the expansion ratios of redundancy removal and direct expansion for each classifier in RL. The average expansion ratios of redundancy removal and direct expansion over the classifiers in RL are 19.9 and 69.9, respectively. The total expansion ratios of redundancy removal and direct expansion over the classifiers in RL are 7.1 and 20.4, respectively. We can see that redundancy removal can significantly reduce the expansion ratio of classifiers.

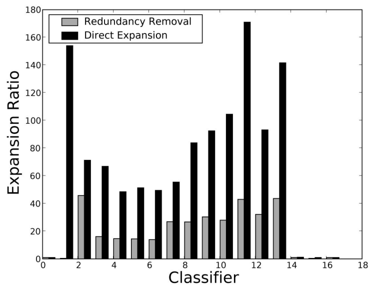


Fig. 13. Expansion ratios of real-life classifiers.

9.3 Effectiveness on Synthetic Classifiers

Classifier rules are considered confidential due to security concerns. Thus, it is difficult to get many real-life classifiers to experiment with. To address this issue and further evaluate the effectiveness of our redundancy removal algorithm, we generated a set of synthetic classifiers, denoted SYN, in the following fashion. Every predicate of a rule in our synthetic classifiers has five fields: source IP address, destination IP address, source port number, destination port number, and protocol type. We first randomly generated a list of values for each field. For IP addresses, we generated a random class-C address; for ports, we generated a random interval; and for protocols, we generated a random protocol number. Given these lists, we generated a list of predicates by taking the cross product of all these lists. We added a final default predicate to our list. Finally, we randomly assigned one of two decisions, accept or deny, to each predicate to make a complete rule.

Redundancy ratio of synthetic classifiers. Fig. 14 shows the distribution of the redundancy ratios achieved by the redundancy removal algorithm over the classifiers in SYN. The average and total redundancy ratios of the classifiers in SYN are 62.4 percent and 76.1 percent, respectively.

Compression ratio of synthetic classifiers. The average and total compression ratios of redundancy removal over the classifiers in SYN are 35.2 percent and 24.4 percent, respectively. Fig. 15 shows the distribution of the compression ratios achieved by the redundancy removal algorithm over the classifiers in SYN.

Expansion ratio of synthetic classifiers. The average expansion ratios of the redundancy removal algorithm and the direct expansion algorithm over the classifiers in SYN are 60.092 and 177.995, respectively. The total expansion ratios of the redundancy removal algorithm and the direct expansion algorithm over the classifiers in SYN are 45.769 and 196.065, respectively. Fig. 16 shows the distribution of

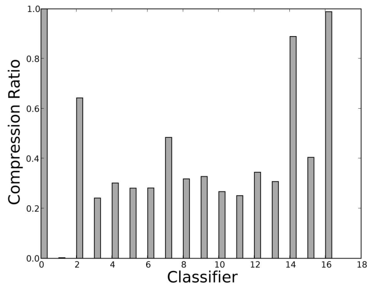


Fig. 12. Compression ratios of real-life classifiers.

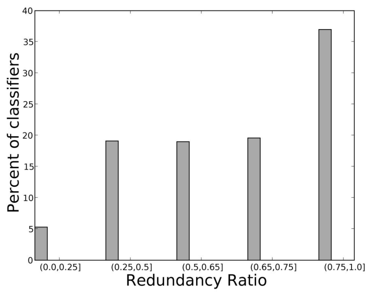


Fig. 14. Distribution of synthetic classifiers by redundancy ratio.

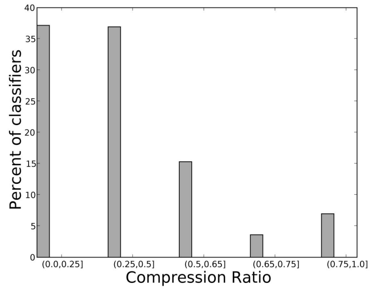


Fig. 15. Distribution of synthetic classifiers by compression ratio.

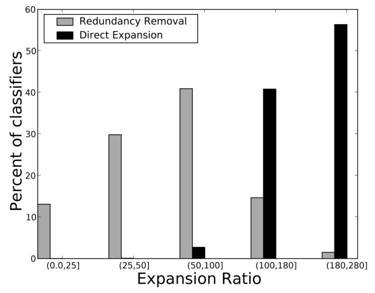


Fig. 16. Distribution of synthetic classifiers by expansion ratio.

TABLE 2
Sample Running Time Data for Real-Life Classifiers

Number of Original Rules	TCAM Razor Running Time (seconds)
42	0.59
87	2.49
661	1.03

expansion ratios achieved by the redundancy removal algorithm and the direct expansion algorithm on the classifiers in SYN.

9.4 Efficiency on Real-Life Classifiers

We implemented the algorithms in this paper in Sun Java JDK 1.4 [20]. The experiments were carried out on a SunBlade 2000 machine running Solaris 9 with a 1-GHz CPU and 1-Gbyte memory.

Table 2 shows the total runtime of our redundancy removal algorithm for three representative classifiers.

9.5 Efficiency on Synthetic Classifiers

Fig. 17 displays the average total runtime of our redundancy removal algorithm on the synthetic classifiers as a function of the number of original rules along with the standard deviation. We can see that our redundancy removal algorithms are efficient. For example, it takes less than 3 seconds to remove all the redundant rules from a classifier that has up to 3,000 rules, and it takes less than 6 seconds to remove all the redundant rules from a classifier that has up to 6,000 rules.

10 An Open Problem: Maximal Redundancy Set

This paper suggests several new research problems. In this section, we briefly describe the maximal redundancy set problem. Addressing this problem is beyond the scope of this paper, but we sketch our thoughts on how one can pursue them in the future.

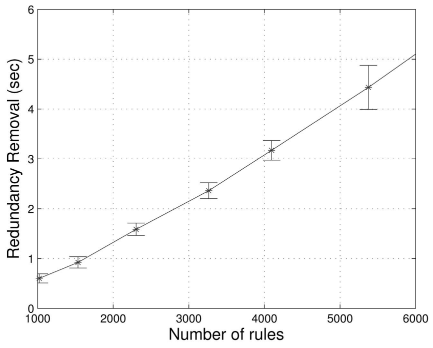


Fig. 17. Average processing time for removing all (both upward and downward) redundant rules versus total number of rules in a classifier.

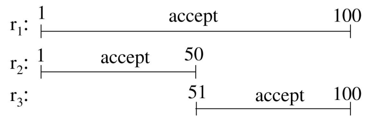


Fig. 18. An illustrative classifier.

As we have noticed, the ordering of detecting and removing redundant rules is critical. Consider the illustrative (although extreme) example in Fig. 18, we can either remove r_1 while keeping r_2 and r_3 or remove r_2 and r_3 while keeping r_1. The order that we used in our redundancy removal, namely, first examining rules from the first to the last for upward redundancy and then examining rules from the last to the first for downward redundancy, has the nice property of guaranteeing that the resulting classifier has no redundant rules.

Next, we describe this issue in a more general way. Given a set of rules s in a classifier f, we use $f - s$ to denote the resulting classifier by removing all the rules in s from f. A set of rules s in a classifier f is redundant if and only if $f - s$ is equivalent to f. A set of rules s is a *complete* set of redundant rules of f if and only if $f - s$ is equivalent to f and $f - s$ has no redundant rules. A set of rules s is a *maximal* set of redundant rules of f if and only if both of the following two conditions are satisfied:

1. s is a complete set of redundant rules of f.
2. For any other complete set s' of redundant rules of f, $|s| \geq |s'|$.

From these definitions, we see that the set of rules removed by our upward and downward redundancy removal algorithms is complete, but the removed set of rules is not guaranteed to be maximal. We are interested in the maximal set of redundant rules in a classifier because we want to reduce the number of rules as much as possible. A straightforward solution could be as follows: given a classifier f, for every subset of rules s, test whether we have $f - s \equiv f$. This equivalence testing procedure is available in [25]. Clearly, this solution is exponential in terms of the number of rules, which could be large. Therefore, the following questions remain open: 1) Is the problem of detecting a maximal set of redundant rules in a classifier NP-complete? 2) If not, what is the polynomial solution?

11 Concluding Remarks

In this paper, we present the first technical study of redundant rules in classifiers. Our contributions are threefold. First, we propose redundancy removal as a new approach to the interval expansion problem of TCAMs. Removing redundant rules directly reduces power consumption of TCAM chips. Our experiments on real-life classifiers showed an average reduction of 58.2 percent on the number of TCAM entries by removing redundant rules. Unlike other interval expansion solutions that require modifying TCAM circuits or packet processing hardware, our approach can be deployed today by network administrators and ISPs to cope with interval expansion. Second, we give a necessary and sufficient condition for identifying redundant rules for the first time. This condition lays the foundation for developing redundancy removal algorithms. Third, we present two tree-based algorithms for removing the two types of redundant rules that we define in this paper, respectively. We formally prove that the resulting classifiers have no redundant rules after running the two algorithms. The experiments that we conducted on both real-life and synthetic classifiers showed that removing all redundant rules from a large classifier with thousands of rules only takes a few seconds using our algorithm.

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

Appendix A

A.1 Proof Sketch for Theorem 1

1. *Equality condition.* We prove this condition by induction. By the definition of matching set and resolving set, we have $M(r_1) = R(r_1, f)$ and $R(r_i, f) = M(r_i) - \bigcup_{j=1}^{i} M(r_j)$. Let us assume the equality condition holds when $i = k$, $(1 \le k \le n)$. When $i = k+1$, we have
$$\begin{aligned}\bigcup_{j=1}^{k+1} M(r_j) &= \bigcup_{j=1}^{k} M(r_j) \cup M(r_{k+1})\\ &= \bigcup_{j=1}^{k} M(r_j) \cup \left(M(r_{k+1}) - \bigcup_{j=1}^{k} M(r_j) \right)\\ &= \bigcup_{j=1}^{k} M(r_j) \cup R(r_{k+1}, f)\\ &= \bigcup_{j=1}^{k} R(r_j, f) \cup R(r_{k+1}, f)\\ &= \bigcup_{j=1}^{k+1} R(r_j, f).\end{aligned}$$
2. *Dependency condition.* By the definition of matching set and resolving set, we have $R(r_i, f) = M(r_i) - \bigcup_{j=1}^{i} M(r_j)$. By the above equality condition, we have $\bigcup_{j=1}^{i-1} M(r_j) = \bigcup_{j=1}^{i-1} R(r_j, f)$. Therefore, $R(r_i, f) = M(r_i) - \bigcup_{j=1}^{i} M(r_j) = M(r_i) - \bigcup_{j=1}^{i-1} R(r_j, f)$.
3. *Determinism condition.* Without loss of generality, we assume that $i < j$. By the dependency condition, we have $R(r_j, f) = M(r_j) - \bigcup_{k=1}^{j-1} R(r_k, f) = M(r_j) - \bigcup_{k=i+1}^{j-1} R(r_k, f) - \bigcup_{k=1}^{i-1} R(r_k, f) - R(r_i, f)$. Therefore, we have $R(r_i, f) \cap R(r_j, f) = \emptyset$.
4. *Comprehensiveness condition.* By the definition of comprehensiveness, we have $\Sigma = \bigcup_{i=1}^{n} M(r_i)$. By the equality condition, we have $\bigcup_{i=1}^{n} M(r_i) = \bigcup_{i=1}^{n} R(r_i, f)$. Therefore, we have $\Sigma = \bigcup_{i=1}^{n} R(r_i, f)$.□

A.2 Proof Sketch for Lemma 1

By Theorem 1, we have $R(r_n, f) = M(r_n) - \bigcup_{j=1}^{n-1} R(r_j, f)$ and $R(r_n, f) = \Sigma - \bigcup_{j=1}^{n-1} R(r_j, f)$. Therefore, $R(r_n, f)$ does not change if we modify $M(r_n)$ to be Σ, i.e., if we modify the predicate of the last rule r_n to be $(F_1 \in D(F_1)) \wedge (F_2 \in D(F_2)) \wedge \cdots \wedge (F_d \in D(F_d))$. □

Acknowledgments

The work of Alex X. Liu is supported in part by the US National Science Foundation (NSF) under Grant CNS-0716407. The work of Mohamed G. Gouda is supported by the NSF under Grant 0520250.

References

[1] Cypress Semiconductor Corp., *Content Addressable Memory,* http://www.cypress.com/, 2008.
[2] *A Guide to Search Engines and Networking Memory,* http://www.linleygroup.com/pdf/NMv4.pdf, 2008.
[3] Integrated Device Technology, Inc., *Content Addressable Memory,* http://www.idt.com/, 2008.
[4] Netlogic Microsystems, *Content Addressable Memory,* http://www.netlogicmicro.com/, 2008.
[5] D.A. Applegate, G. Calinescu, D.S. Johnson, H. Karloff, K. Ligett, and J. Wang, "Compressing Rectilinear Pictures and Minimizing Access Control Lists," *Proc. 18th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA '07),* Jan. 2007.
[6] A. Bremler-Barr and D. Hendler, "Space-Efficient TCAM-Based Classification Using Gray Coding," *Proc. IEEE INFOCOM '07,* May 2007.
[7] H. Che, Z. Wang, K. Zheng, and B. Liu, "DRES: Dynamic Range Encoding Scheme for TCAM Coprocessors," *IEEE Trans. Computers,* vol. 57, no. 7, pp. 902-915, 2008.
[8] Q. Dong, S. Banerjee, J. Wang, D. Agrawal, and A. Shukla, "Packet Classifiers in Ternary CAMs Can Be Smaller," *Proc. ACM SIGMETRICS '06,* pp. 311-322, 2006.
[9] R. Draves, C. King, S. Venkatachary, and B. Zill, "Constructing Optimal IP Routing Tables," *Proc. IEEE INFOCOM '99,* pp. 88-97, 1999.
[10] 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,* vol. 20, no. 3, pp. 263-270, 2001.
[11] M. Gouda, A.X. Liu, and M. Jafry, "Verification of Distributed Firewalls," *Proc. IEEE Global Comm. Conf. (GLOBECOM),* 2008.
[12] M.G. Gouda and A.X. Liu, "Firewall Design: Consistency, Completeness and Compactness," *Proc. 24th IEEE Int'l Conf. Distributed Computing Systems (ICDCS '04),* pp. 320-327, Mar. 2004.
[13] M.G. Gouda and A.X. Liu, "A Model of Stateful Firewalls and its Properties," *Proc. IEEE Int'l Conf. Dependable Systems and Networks (DSN '05),* pp. 320-327, June 2005.
[14] M.G. Gouda and A.X. Liu, "Structured Firewall Design," *Computer Networks J.,* vol. 51, no. 4, pp. 1106-1120, Mar. 2007.
[15] P. Gupta, "Algorithms for Routing Lookups and Packet Classification," PhD thesis, Stanford Univ., 2000.

[16] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields," *Proc. ACM SIGCOMM '99,* pp. 147-160, 1999.

[17] P. Gupta and N. McKeown, "Packet Classification Using Hierarchical Intelligent Cuttings," *Proc. Seventh Symp. High-Performance Interconnects (Hot Interconnects '99),* Aug. 1999.

[18] P. Gupta and N. McKeown, "Algorithms for Packet Classification," *IEEE Network,* vol. 15, no. 2, pp. 24-32, 2001.

[19] J. Hwang, T. Xie, F. Chen, and A.X. Liu, "Systematic Structural Testing of Firewall Policies," *Proc. 27th IEEE Int'l Symp. Reliable Distributed Systems (SRDS),* 2008.

[20] *Java,* http://java.sun.com/, Sept. 2004.

[21] S. Kamara, S. Fahmy, E. Schultz, F. Kerschbaum, and M. Frantzen, "Analysis of Vulnerabilities in Internet Firewalls," *Computers and Security,* vol. 22, no. 3, pp. 214-232, 2003.

[22] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for Advanced Packet Classification with Ternary CAMs," *Proc. ACM SIGCOMM '05,* pp. 193-204, Aug. 2005.

[23] A.X. Liu, "Change-Impact Analysis of Firewall Policies," *Proc. 12th European Symp. Research Computer Security (ESORICS '07),* pp. 155-170, Sept. 2007.

[24] A.X. Liu, "Firewall Policy Verification and Troubleshooting," *Proc. IEEE Int'l Conf. Comm. (ICC '08),* May 2008.

[25] A.X. Liu and M.G. Gouda, "Diverse Firewall Design," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '04),* pp. 595-604, June 2004.

[26] A.X. Liu and M.G. Gouda, "Complete Redundancy Detection in Firewalls," *Proc. 19th Ann. IFIP Conf. Data and Applications Security,* pp. 196-209, Aug. 2005.

[27] A.X. Liu and M.G. Gouda, "Diverse Firewall Design," *IEEE Trans. Parallel and Distributed Systems,* vol. 19, no. 8, 2008.

[28] A.X. Liu, M.G. Gouda, H.H. Ma, and A.H. Ngu, "Firewall Queries," *Proc. Eighth Int'l Conf. Principles of Distributed Systems (OPODIS '04),* T. Higashino, ed., pp. 124-139, Dec. 2004.

[29] A.X. Liu, C.R. Meiners, and Y. Zhou, "All-Match Based Complete Redundancy Removal for Packet Classifiers in TCAMs," *Proc. IEEE INFOCOM '08,* Apr. 2008.

[30] A.X. Liu, E. Torng, and C. Meiners, "Firewall Compressor: An Algorithm for Minimizing Firewall Policies," *Proc. IEEE INFOCOM '08,* Apr. 2008.

[31] H. Liu, "Efficient Mapping of Range Classifier into Ternary-CAM," *Proc. 14th Symp. High-Performance Interconnects (Hot Interconnects '02),* pp. 95-100, 2002.

[32] C.R. Meiners, A.X. Liu, and E. Torng, "TCAM Razor: A Systematic Approach towards Minimizing Packet Classifiers in TCAMs," *Proc. 15th IEEE Conf. Network Protocols (ICNP '07),* pp. 266-275, Oct. 2007.

[33] M.H. Overmars and A.F. van der Stappen, "Range Searching and Point Location among Fat Objects," *J. Algorithms,* vol. 21, no. 3, pp. 629-656.

[34] D. Pao, P. Zhou, B. Liu, and X. Zhang, "Enhanced Prefix Inclusion Coding Filter-Encoding Algorithm for Packet Classification with Ternary Content Addressable Memory," *IET Computers & Digital Techniques,* vol. 1, pp. 572-580, Apr. 2007.

[35] E. Spitznagel, D. Taylor, and J. Turner, "Packet Classification Using Extended TCAMs," *Proc. 11th IEEE Int'l Conf. Network Protocols (ICNP '03),* pp. 120-131, Nov. 2003.

[36] S. Suri, T. Sandholm, and P. Warkhede, "Compressing Two-Dimensional Routing Tables," *Algorithmica,* vol. 35, pp. 287-300, 2003.

[37] D.E. Taylor, "Survey & Taxonomy of Packet Classification Techniques," *ACM Computing Surveys,* vol. 37, no. 3, pp. 238-275, 2005.

[38] J. van Lunteren and T. Engbersen, "Fast and Scalable Packet Classification," *IEEE J. Selected Areas in Comm.,* vol. 21, no. 4, pp. 560-571, 2003.

[39] T.Y.C. Woo, "A Modular Approach to Packet Classification: Algorithms and Results," *Proc. IEEE INFOCOM '00,* pp. 1213-1222, 2000.

[40] F. Yu, T.V. Lakshman, M.A. Motoyama, and R.H. Katz, "SSA: A Power and Memory Efficient Scheme to Multi-Match Packet Classification," *Proc. Symp. Architectures for Networking and Comm. Systems (ANCS '05),* pp. 105-113, Oct. 2005.

[41] K. Zheng, H. Che, Z. Wang, B. Liu, and X. Zhang, "DPPC-RE: TCAM-Based Distributed Parallel Packet Classification with Range Encoding," *IEEE Trans. Computers,* vol. 55, no. 8, pp. 947-961, 2006.

[42] K. Zheng, C. Hu, H. Lu, and B. Liu, "A TCAM-Based Distributed Parallel IP Lookup Scheme and Performance Analysis," *IEEE/ACM Trans. Networking,* vol. 14, no. 4, pp. 863-875, 2006.

Alex X. Liu received the PhD degree in computer science from the University of Texas at Austin in 2006. He is currently an assistant professor in the Department of Computer Science and Engineering, Michigan State University, East Lansing. He received the 2004 IEEE/IFIP William C. Carter Award, the 2004 National Outstanding Overseas Students Award sponsored by the Ministry of Education of China, the 2005 George H. Mitchell Award for Excellence in Graduate Research in the University of Texas at Austin, and the 2005 James C. Browne Outstanding Graduate Student Fellowship in the University of Texas at Austin. His research interests include computer and network security, dependable and high-assurance computing, applied cryptography, computer networks, operating systems, and distributed computing. He is a member of the IEEE.

Mohamed G. Gouda was born in Egypt. He received BSc degree in engineering and the BSc degree in mathematics from Cairo University, the MA in mathematics from York University, and the master's and PhD degrees in computer science from the University of Waterloo. He worked for the Honeywell Corporate Technology Center, Minneapolis, from 1977 to 1980. In 1980, he joined the University of Texas at Austin, where he currently holds the Mike A. Myers Centennial Professorship in Computer Sciences. He is the 1993 winner of the Kuwait Award in Basic Sciences. He received the 2001 IEEE Communication Society William R. Bennet Best Paper Award for his paper "Secure Group Communications Using Key Graphs," coauthored with C.K. Wong and S.S. Lam and published in the *IEEE/ACM Transactions on Networking* (vol. 8, no. 1, pp. 16-30). In 2004, he received the William C. Carter Award for his paper "Diverse Firewall Design," coauthored with Alex X. Liu and published in the *Proceedings of the International Conference on Dependable Systems and Networks.* He is a member of the IEEE.