Removing Redundancy from Packet Classifiers

Alex X. Liu   Mohamed G. Gouda

TR-04-26
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188, U.S.A.
{alex, gouda}@cs.utexas.edu

June 4, 2004

Abstract

Packet classification is the core mechanism that enables many networking services such as firewall access control and traffic accounting. Reducing memory space for packet classification algorithms is of paramount importance because a packet classifier must use very limited on-chip cache to store complex data structures. This paper proposes the first ever scheme that can significantly reduce memory space for all packet classification algorithms. The scheme is to remove all redundant rules in a packet classifier before a classification algorithm starts building data structures. By removing redundant rules, we can save more than 73% of memory for a packet classifier that examines eight packet fields. In this paper, we categorize redundant rules into upward redundant rules and downward redundant rules. We give a necessary and sufficient condition for identifying each type of redundant rule. We present two efficient algorithms for detecting and removing the two types of redundant rules respectively. The two algorithms make use of a graph model of packet classifiers, called packet decision diagrams. The experimental results shows that our algorithms are very efficient.

1. Introduction

Most routers on the Internet have packet classification capabilities. Packet classification is the core mechanism that enables routers to perform many network services, such as routing [9], active networking [8], firewall access control [3], quality of service [1], differential service [2], etc. A packet classifier maps each packet to a decision based on a sequence of rules. 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. The possible decisions to which a packet classifier can map a packet are application specific. For example, the possible decision to which a packet is mapped by a packet classifier that is used as a firewall can be either *accept* or *discard*. Each rule in a packet classifier is of the form $\langle predicate \rangle \rightarrow \langle decision \rangle$ where the $\langle predicate \rangle$ is a boolean expression over some packet fields. A packet *matches* a rule iff the packet satisfies the predicate of the rule. A packet may match more than one rule in a packet classifier. Therefore, a packet classifier maps each packet to the decision of the first (i.e., highest priority) rule that the packet matches.

A packet classifier may have redundant rules. A rule in a packet classifier is redundant iff removing the rule does not change the decision of the packet classifier for each packet. For example, consider the simple packet classifier in Figure 1. This packet classifier consists of four rules r_1 through r_4, where each rule only checks one packet field F_1 whose domain of values is $[1, 100]$.

r_1 : $F_1 \in [1,\ 50] \rightarrow accept$
r_2 : $F_1 \in [40,\ 90] \rightarrow discard$
r_3 : $F_1 \in [30,\ 60] \rightarrow accept$
r_4 : $F_1 \in [51,\ 100] \rightarrow discard$

Figure 1. A simple packet classifier

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

1. Rule r_3 is redundant. This is because the first matching rule for all packets where $F_1 \in [30, 50]$ is r_1, and the first matching rule for all packets where

$F_1 \in [51, 60]$ is r_2. Therefore, there are no packets whose first matching rule is r_3. We call r_3 an upward redundant rule. A rule r in a packet classifier is *upward redundant* iff there are no packets whose first matching rule is r. Geometrically, a rule is upward redundant in a packet classifier if the rule is overlayed by some rules listed above it.

2. Rule r_2 becomes redundant after r_3 is removed. Note that r_2 is the first matching rule for all packets where $F_1 \in [51, 90]$. However, if both r_2 and r_3 are removed, the first matching rule for all those packets becomes r_4 instead of r_2. This is acceptable since both r_2 and r_4 have the same decision. We call r_2 a downward redundant rule. A rule r in a packet classifier, where no rule is upward redundant, is *downward redundant* iff for each packet, whose first matching rule is r, the first matching rule below r has the same decision as r.

Gupta identified two special types of redundant rules in his PhD thesis [4], namely backward redundant rules and forward redundant rules, by studying 793 packet classifiers from 101 different Internet Service Providers and enterprise networks with a total of $41,505$ rules. A rule r in a packet classifier is backward redundant iff there exists another rule r' listed above r such that all packets that match r also match r'. In [4], Gupta observed that on average 7.8% of the rules in a packet classifier are backward redundant. Clearly, a backward redundant rule is an upward redundant rule, but not vice versa. For example, rule r_2 in Figure 1 is upward redundant, but not backward redundant.

A rule r in a packet classifier is forward redundant iff there exists another rule r' listed below r such that the following three conditions hold: (1) all packets that match r also match r', (2) r and r' have the same decision, (3) for each rule r'' listed between r and r', either r and r'' have the same decision, or no packet matches both r and r''. In [4], Gupta observed that on average 7.2% of the rules in a packet classifier are forward redundant. Clearly, a forward redundant rule is a downward redundant rule, but not vice versa. For example, rule r_2 in Figure 1, assuming r_3 has been removed previously, is downward redundant, but not forward redundant.

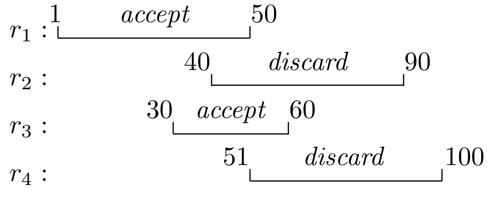


Figure 2. Geometric representation of the rules in Figure 1

In [3], Gouda and Liu identified redundancy in the sequence of rules generated from a firewall decision diagram. However, the algorithms for removing redundant rules in [3] are applicable only to the rules generated from a firewall decision diagram. By contrast, the algorithms presented in the current paper can be applied to any packet classifier.

Previous work on packet classification has focused on developing efficient classification algorithms (e.g., [11, 12, 13]). A packet classification algorithm builds a data structure from the sequence of rules in a packet classifier, and uses this data structure to search for the decision of the first rule that a packet matches. The design goals of all these algorithms are to reduce the classification time and space. The classification time is the average processing time that a packet classification algorithm needs to find the decision for a packet. The classification space is the amount of memory needed to store the (usually large) data structures of a packet classification algorithm. Reducing classification space for packet classification algorithms is of paramount importance because small classification space enables the use of very limited on-chip cache to store the data structure of a packet classification algorithm. In other words, reducing classification space has significant impact on reducing classification time.

In this paper, we propose to remove all redundant rules from a packet classifier before a packet classification algorithm starts building its data structure from the rules. We give a necessary and sufficient condition for identifying all redundant rules. We categorize redundant rules into upward redundant rules and downward redundant rules. We present two efficient graph based algorithms for removing these two types of redundant rules. The experimental results show that these two algorithms are very efficient.

Removing the redundant rules from a packet classifier has the following three main merits:

1. **Complement classification algorithms**: The algorithms presented in this paper for removing redundant rules are not intended to replace any of the previous (or future) classification algorithms. Rather, it complements these algorithms since redundancy removal can be viewed as a preprocessing procedure for each of these classification algorithms.

2. **Reduce classification space**: Based on the complexity bounds from computational geometry in [10], the fastest packet classification algorithm needs $O(n^d)$ classification space (and $O(\log n)$ classification time), where n is the total number of rules and d ($d > 3$) is the total number of packet

fields that the classifier examines for each packet. Most fast packet classification algorithms, such as Recursive Flow Classification [5], have $O(\log n)$ complexity with $O(n^d)$ memory space. It has been observed in [4] that on average a packet classifier has 15% redundant rules (based on Gupta's definition of redundant rules). For a packet classifier with 15% redundant rules, Figure 3 shows the percentage of classification space that is saved by removing redundant rules versus the number of fields that a packet classifier checks. From this figure, we see that removing redundant rules from a packet classifier saves about 48% memory space when $d = 4$ and saves about 80% memory space when $d = 10$.

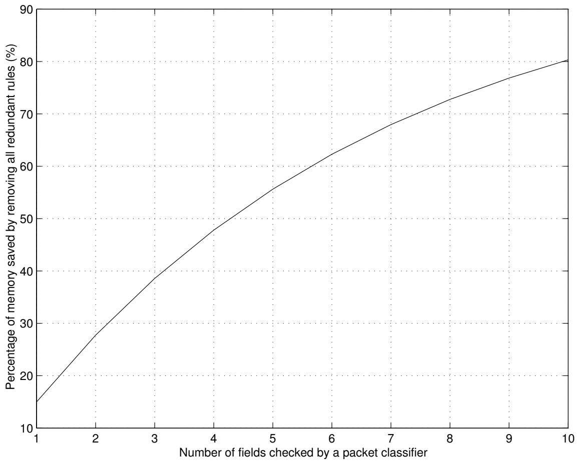


Figure 3. Number of fields vs. Memory Saved

3. **Reduce classification time**: It has been observed in [7] that reducing the amount of overlapping among rules or reducing the total number of rules reduces classification time. Two rules *overlap* iff there is at least one packet that can match both rules. Redundancy in a packet classifier is caused by the overlapping of rules. Each redundant rule overlaps with some other rules. By removing redundant rules, while other non-redundant rules remain unchanged, both the amount of overlapping of rules and the total number of rules are reduced. Therefore, removing redundant rules directly reduces classification time.

The rest of this paper is organized as follows. We give a necessary and sufficient condition for identifying redundant rules in Section 2. In Section 3, we introduce Packet Decision Diagrams, which will be used as the core data structure for redundancy removal algorithms. The upward and downward redundancy removal algorithms are presented in Section 4 and 5. The experimental results are shown in Section 6. We give concluding remarks in Section 7.

2. Redundancy of Packet Classifiers

We 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 $[0, 2^{32}-1]$. We use Σ to denote the set of all packets over fields $F_1, F_2, \cdots, F_d$. It follows that Σ is a finite set and $|\Sigma| = |D(F_1)| \times \cdots \times |D(F_n)|$.

A *packet classifier*, over the fields $F_1, \cdots, F_d$ and whose decision set is DS, is a sequence of rules, and each rule is of the following format:

$$(F_1 \in S_1) \wedge (F_2 \in S_2) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle decision \rangle$$

where each S_i is a nonempty subset of $D(F_i)$ and $\langle decision \rangle$ is an element of DS. A packet $(p_1, \cdots, p_d)$ *matches* a rule $(F_1 \in S_1) \wedge (F_2 \in S_2) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle decision \rangle$ iff the following condition holds:

$$(p_1 \in S_1) \wedge (p_2 \in S_2) \wedge \cdots \wedge (p_d \in S_d)$$

For simplicity, in the rest of this paper, we assume that all packets and all packet classifiers are over the fields $F_1, F_2, \cdots, F_d$, if not otherwise specified.

Next we define two important concepts: matching set and resolving set. Consider a packet classifier f that consists of n rules $\langle r_1, r_2, \cdots, r_n \rangle$. The *matching set* of a rule r_i in this packet classifier is the set of all packets that match r_i. The *resolving set* of a rule r_i in this packet classifier is the set of all packets that match r, but do not match any r_j that $j < i$. For example, consider the rule r_2 in Figure 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 $[51, 90]$. The matching set of a rule r_i is denoted $M(r_i)$, and the resolving set of a rule r_i is denoted $R(r_i, f)$. Note that the matching set of a rule depends only on the rule itself, while the resolving set of a rule depends both the rule itself and all the rules listed above it in a packet classifier.

From the definition of $M(r_i)$ and $R(r_i, f)$, we have

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

Therefore, we have the following theorem:

Theorem 1 Let f be any packet classifier that consists of n rules: $\langle r_1, r_2, \cdots, r_n \rangle$. For each i, $1 \leq i \leq n$, we have:

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

□

A sequence of rules $\langle r_1, r_2, \cdots, r_n \rangle$ is *comprehensive* iff for any packet p, there is at least one rule that matches p in the sequence. A sequence of rules must be comprehensive for it to serve as a packet classifier. From now on, we assume each packet classifier is comprehensive. Therefore, we have the following theorem:

Theorem 2 Let f be any packet classifier that consists of n rules: $\langle r_1, r_2, \cdots, r_n \rangle$. The following two conditions hold:

1. Determinism: $R(r_i, f) \cap R(r_j, f) = \emptyset \ (i \neq j)$
2. Comprehensiveness: $\bigcup_{i=1}^{n} R(r_i, f) = \Sigma$ □

We use $f(p)$ to denote the decision to which a packet classifier f maps a packet p. Two packet classifiers 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.

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

Theorem 3 Let f be any packet classifier that consists of n rules: $\langle r_1, r_2, \cdots, r_n \rangle$. 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 packet 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.

Proof Sketch: By Theorem 1, we have $R(r_n, f) = M(r_n) - \bigcup_{j=1}^{n-1} R(r_j, f)$, and by Theorem 2, we have $R(r_n, f) = \Sigma - \bigcup_{j=1}^{n-1} R(r_j, f)$. So $R(r_n, f)$ 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 $(F_1 \in D(F_1)) \wedge (F_2 \in D(F_2)) \wedge \cdots \wedge (F_d \in D(F_d))$. □

By modifying rule r_n in this way, any postfix of a packet classifier is comprehensive, i.e., if $\langle r_1, r_2, \cdots, r_n \rangle$ is comprehensive, then $\langle r_i, r_{i+1}, \cdots, r_n \rangle$ is comprehensive for each i, $1 \leq i \leq n$. In the rest of this paper, we assume the predicate of the last rule in a packet classifier is $(F_1 \in D(F_1)) \wedge (F_2 \in D(F_2)) \wedge \cdots \wedge (F_d \in D(F_d))$.

Redundant rules are defined as follows.

Definition 1 A rule r is *redundant* in a packet classifier f iff the resulting packet classifier f' after removing rule r is equivalent to f.

The following theorem shows a necessary and sufficient condition for identifying redundant rules. The correctness of this theorem can be proven in a straightforward way by the above discussion. Note that we use the notation $\langle r_{i+1}, r_{i+2}, \cdots, r_n \rangle(p)$ to denote the decision to which the packet classifier $\langle r_{i+1}, r_{i+2}, \cdots, r_n \rangle$ maps the packet p.

Theorem 4 (Redundancy Theorem) Let f be any packet classifier that consists of n rules: $\langle r_1, r_2, \cdots, r_n \rangle$. A rule r_i is redundant in f iff one of the following two conditions holds:

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

By the redundancy theorem, we categorize all redundant rules into upward and downward redundant rules.

Definition 2 A rule that satisfies the first condition in the redundancy theorem is called an *upward redundant rule*, whereas a rule that satisfies the second condition in the redundancy theorem is called a *downward redundant rule*.

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

2.1. Upward/Downward Redundancy vs. Backward/Forward Redundancy

Next we argue that the backward redundant rules defined by Gupta in [4] form a (sometimes proper) subset of upward redundant rules, and similarly the forward redundant rules defined in [4] form a (sometimes proper) subset of downward redundant rules. In other words, the classification of redundancy in [4] is not as complete as our classification of redundancy in this paper. Therefore, more rules can be removed from a packet classifier using our definition (of upward and downward redundancy) without changing the function of the packet classifier.

Let f be any packet classifier that consists of n rules: $\langle r_1, r_2, \cdots, r_n \rangle$.

1. *Upward redundancy vs. backward redundancy*:
   By Gupta's definition, a rule r_i is backward redundant in f iff there exists k, $1 \leq k < i$, such that $M(r_i) \subseteq M(r_k)$. Clearly, if there exists such k for r_i, then $R(r_i, f) = M(r_i) - \bigcup_{j=1}^{i-1} M(r_j) = \emptyset$; therefore, r_i is upward redundant. However, if $R(r_i, f) = \emptyset$, such k may not exist. As an example, in the packet classifier in Figure 1, rule r_3 is upward redundant, but not backward redundant. Thus r_3 can be removed based on our definition,

but it cannot be removed based on Gupta's definition.

2. *Downward redundancy vs. forward redundancy*: By Gupta's definition, a rule r_i is forward redundant iff there exists k, $i < k \leq n$, such that the following three conditions hold: (1) $M(r_i) \subseteq M(r_k)$, (2) r_i and r_k have the same decision, (3) for any j that $i < j < k$, either $M(r_i) \cap M(r_j) = \emptyset$ or r_i and r_j have the same decision. Clearly, if there exists such k for r_i, then for any p that $p \in R(r_i, f)$, the decision $\langle r_{i+1}, r_{i+2}, \cdots, r_n\rangle(p)$ is the same as the decision of r_i; therefore, r_i is downward redundant. However, a rule may be downward redundant even if there is no such k. As an example, in the packet classifier that results from the classifier in Figure 1 after r_3 is removed, rule r_2 is downward redundant, but not forward redundant. Thus r_2 can be removed based on our definition, but it cannot be removed based on Gupta's definition.

3. Packet Decision Diagrams and Rules

In [3], Gouda and Liu presented Firewall Decision Diagrams as a useful notation for specifying firewalls. In this paper, we extend these diagrams to specify packet classifiers; therefore, we call the extended decision diagrams Packet Decision Diagrams. Later we show that Packet Decisions Diagrams play an important role in our redundancy removal algorithms.

Definition 3 A *Packet Decision Diagram* (PDD) f with a decision set DS and over fields $F_1, \cdots, F_d$ is an acyclic and directed graph that has the following five properties:

1. There is exactly one node in f that has no incoming edges and is called the *root* of f. The nodes in f that have no outgoing edges are called *terminal* nodes of f.
2. Each node v in f has a label, denoted $F(v)$, such that

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

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

Figure 4 shows an example of a PDD 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 of this paper, we employ the decision set $\{a, d\}$, where "a" represents "accept" and "d" represents "discard".

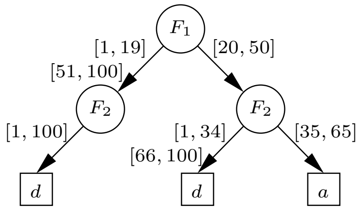


Figure 4. A packet decision diagram

A decision path in a PDD f is represented by $(v_1 e_1 \cdots v_k e_k v_{k+1})$ 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 $(v_1 e_1 \cdots v_k e_k v_{k+1})$ in a PDD defines the following rule:

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

where

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

For a PDD 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 PDD 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 a PDD f maps a packet p. A PDD f and a sequence of rules f' are equivalent, denoted $f \equiv f'$, iff for any packet p, the condition $f(p) = f'(p)$ holds.

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

Given a sequence of rules, in section 4 we will see that an equivalent PDD 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 PDD. A *standard* PDD is a special type of PDD where the following two additional conditions hold:

1. each node has at most one incoming edge (i.e., a standard PDD is of a tree structure),
2. each decision path contains d nonterminal nodes, and the i-th node is labelled F_i for each i that $1 \leq i \leq d$ (i.e., each decision path in a standard PDD is of the form $(v_1 e_1 v_2 e_2 \cdots v_d e_d v_{d+1})$ where $F(v_i) = F_i$ for each i that $1 \leq i \leq d$).

An example of a standard PDD is in Figure 4.

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

We use S_f to denote the set of all the rules defined by all the decision paths in a partial PDD f. For any packet p that $p \in \bigcup_{r \in S_f} M(r)$, 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 PDD f and a sequence of rules $\langle r_1, r_2, \cdots, r_k \rangle$ that may be not comprehensive, we say f is *equivalent* to $\langle r_1, r_2, \cdots, r_k \rangle$ iff 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 $\langle r_1, r_2, \cdots, r_k \rangle$.

An example of a partial PDD is in Figure 8.

4. Removing Upward Redundancy

In this section, we discuss how to remove upward redundant rules. By definition, a rule is upward redundant iff its resolving set is empty. Therefore, in order to remove all upward redundant rules from a packet classifier, we need to calculate resolving set for each rule in the packet classifier. The resolving set of each rule is calculated by its effective rule set. An effective rule set of a rule r in a packet classifier f is a set of non-overlapping 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 4 Let r be a rule in a packet classifier f. A set of rules $\{r'_1, r'_2, \cdots, r'_k\}$ is an *effective rule set* of r iff the following three conditions hold:

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

For example, consider the packet classifier in Figure 1. Then, $\{F_1 \in [1, 50] \rightarrow \textit{accept}\}$ is an effective rule set of rule r_1, $\{F_1 \in [51, 90] \rightarrow \textit{discard}\}$ is an effective rule set of rule r_2, $\emptyset$ is an effective rule set of rule r_3, and $\{F_1 \in [91, 100] \rightarrow \textit{discard}\}$ is an effective rule set of rule r_4. Clearly, once we obtain an effective rule set of a rule r in a packet 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 an 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 5 straightforwardly follows from the above discussion.

Theorem 5 A rule r is upward redundant in a packet classifier iff an effective rule set of r is empty. □

Based on Theorem 5, the basic idea of our upward redundancy removal algorithm is as follows: given a packet classifier $\langle r_1, r_2, \cdots, r_n \rangle$, 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 packet classifier?

An effective rule set for each rule in a packet classifier is calculated with the help of partial PDDs. Consider a packet classifier that consists of n rules $\langle r_1, r_2, \cdots, r_n \rangle$. Our upward redundancy removal algorithm first builds a partial PDD, 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 PDD f_1 to another partial PDD, 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 PDD that we constructed from the rule sequence $\langle r_1, r_2, \cdots, r_i \rangle$, 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 PDD f_i to the partial PDD f_{i+1}, and the calculation of E_{i+1}. Consider the packet classifier in Figure 5 with the decision set $\{a, d\}$ and over fields F_1 and F_2, where $D(F_1) = D(F_2) = [1, 100]$. Figure 6 shows the geometric representation of this packet classifier, where each rule is represented by a rectangle. From Figure 6, 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.

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

Figure 5. A packet classifier of 4 rules

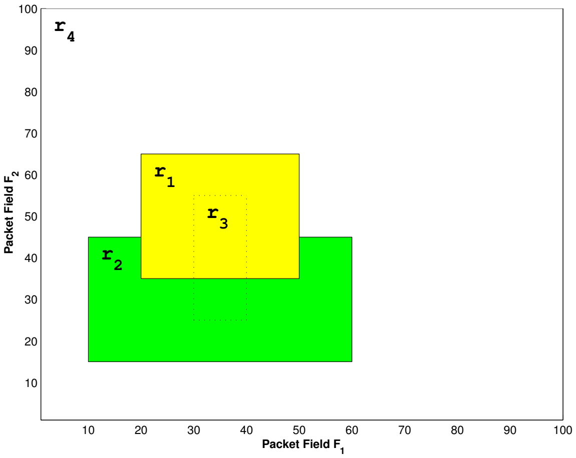
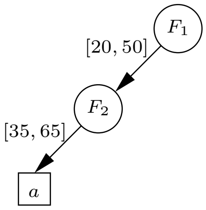


Figure 6. Geometric representation of the rules in Figure 5

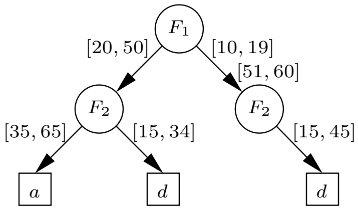
Figure 7 shows a partial PDD f_1 that is equivalent to $\langle r_1 \rangle$ and an effective rule set E_1 of rule r_1. In this figure, we use v_1 to denote the node with label F_1, e_1 to denote the edge with label $[20, 50]$, and v_2 to denote the node with label F_2.



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

Figure 7. Partial PDD f_1 and an effective rule set E_1 of rule r_1 in Figure 5

Now we show how to append rule r_2 to f_1 in order to get a partial PDD 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 $[20, 50]$ labelled on the outgoing edge of v_1. Since $[10, 60] - [20, 50] = [10, 19] \cup [51, 60]$, r_2 is the first matching rule for all 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 $F_1 \in [20, 50]$. So 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. Similarly 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 PDD f_2 and an effective rule set E_2 of rule r_2 is shown in Figure 8, where E_2 consists of the two rules defined by the two new edges e and e' that we add to the partial PDD f_1 in Figure 7.



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

Figure 8. Partial PDD f_2 and an effective rule set E_2 of rule r_2 in Figure 5

Let f be any packet classifier that consists of n rules: $\langle r_1, r_2, \cdots, r_n \rangle$. A partial PDD that is equivalent to $\langle r_1 \rangle$ is easy to construct. Assuming 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 PDD 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 $1 \le i \le d$ and $F(v_{d+1}) = \langle decision \rangle$, is equivalent to $\langle r_1 \rangle$. We denote this partial PDD 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 PDD f_i that is equivalent to the sequence $\langle r_1, r_2, \cdots, r_i \rangle$, and calculated an effective rule set for each of these i rules. Let v be the root of f_i, and assume v has k outgoing edges $e_1, e_2, \cdots, e_k$. 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 PDD f_i to a partial PDD, denoted f_{i+1}, that is equivalent to the sequence $\langle r_1, r_2, \cdots, r_i, r_{i+1}\rangle$, 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 $(F_2 \in S_2) \wedge (F_3 \in S_3) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle \mathit{decision} \rangle$. The rule r, $(F_1 \in S_1 - (I(e_1) \cup I(e_2) \cup \cdots \cup I(e_k))) \wedge (F_2 \in S_2) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle \mathit{decision} \rangle$, defined by the decision path containing the new edge e_{k+1} has the property $M(r) \subseteq R(r_{i+1}, f)$. Therefore, we add rule r to E_i.

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

1. $S_1 \cap I(e_j) = \emptyset$: In this case, we skip edge e_j because any packet whose value of field F_1 is in set $I(e_j)$ doesn't match r_{i+1}.
2. $S_1 \cap I(e_j) = I(e_j)$: In this case, for a packet p whose value of field F_1 is in set $I(e_j)$, the first rule that p matches may be one of the first i rules, and may be rule r_{i+1}. So we append $(F_2 \in S_2) \wedge (F_3 \in S_3) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle \mathit{decision} \rangle$ 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 PDD 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 PDD f_i is transformed to f_{i+1}, the rules in E_{i+1} satisfy the following three 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, (2) no overlapping among these rules by the consistency properties of a partial PDD, (3) 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 packet classifier in Figure 5, we get an effective rule set for each rule as shown in Figure 9. Note that $E_3 = \emptyset$, which means that rule r_3 is upward redundant, therefore r_3 is removed.

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

Figure 9. Effective rule sets calculated for the packet classifier in Figure 5

The pseudocode for removing upward redundant rules is in Figure 10. In the algorithm, we use $e.t$ to denote the node that edge e points to.

```
Upward Redundancy Removal Algorithm
input  : A packet 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( $e.t$, $i$, $(F_{j+1} \in S_{j+1}) \wedge \cdots$
                               $\wedge (F_d \in S_d) \rightarrow \langle decision \rangle$ );
```

Figure 10. Upward Redundancy Removal Algorithm

5. Removing Downward Redundancy

One particular advantage of detecting and removing upward redundant rules before detecting and removing downward redundant rules in a packet classifier is that an effective rule set for each rule is calculated by the upward redundancy removal algorithm; therefore, we can use the effective rule set of a rule to check whether the rule is downward redundant. The effective rule set E_i calculated for rule r_i in a packet classifier f is important 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 6 Let f be any packet classifier that consists of n rules: $\langle r_1, r_2, \cdots, r_n \rangle$. Let f_i $(2 \leq i \leq n)$ be a standard PDD that is equivalent to the sequence of rules $\langle r_i, r_{i+1}, \cdots, r_n \rangle$. The rule r_{i-1} with an effective rule set E_{i-1} is downward redundant in f iff for each rule r in E_{i-1} and for each decision path $(v_1 e_1 v_2 e_2 \cdots v_d e_d v_{d+1})$ 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 $\langle r_i, r_{i+1}, \cdots, r_n \rangle$ is comprehensive, there exists a standard PDD that is equivalent to this sequence of rules. By the redundancy theorem, rule r_{i-1} is downward redundant iff for each rule r in E_{i-1} and for any p that $p \in M(r)$, $\langle r_i, r_{i+1}, \cdots, r_n \rangle(p)$ is the same as the decision of r. Therefore, Theorem 6 follows. □

Now we consider how to construct a standard PDD f_i, $2 \leq i \leq n$, that is equivalent to the sequence of rules $\langle r_i, r_{i+1}, \cdots, r_n \rangle$. The standard PDD 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 PDD f_i that is equivalent to the sequence of rules $\langle r_i, r_{i+1}, \cdots, r_n \rangle$. First, we check whether rule r_{i-1} is downward redundant by Theorem 6. If rule r_{i-1} is downward redundant, then we remove r_i, rename the standard PDD 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 PDD f_i such that the resulting diagram is a standard PDD, denoted f_{i-1}, that is equivalent to the sequence of rules $\langle r_{i-1}, r_i, \cdots, r_n \rangle$. This procedure of transforming a standard PDD by appending a rule is similar to the procedure of transforming a partial PDD 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 in Figure 11.

Applying our downward redundancy removal algorithm to the packet classifier in Figure 5, assuming r_3 has been removed, rule r_2 is detected to be downward redundant, therefore r_2 is removed. The standard PDD in Figure 4 is the resulting standard PDD by appending rule r_1 to the standard PDD that is equivalent to $\langle r_4 \rangle$.

```
Downward Redundancy Removal Algorithm
input  : A packet 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;
```

Figure 11. Downward Redundancy Removal Algorithm

6. Experimental Results

In this section, we evaluate the efficiency of the upward and downward redundancy removal algorithms. In the absence of publicly available packet classifiers, we create synthetic packet classifiers that embody the important characteristics of real-life packet classifiers that have been discovered so far in [4, 11].

We implemented the algorithms in this paper in SUN Java JDK 1.4 [6]. The experiments were carried out on one SunBlade 2000 machine running Solaris 9 with 1Ghz CPU and 1 GB memory. The average processing time for removing all upward and downward redundant rules from a packet classifier versus the total number of rules in the packet classifier is shown in Figure 12. 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 packet classifier that has up to 3000 rules, and it takes less than 6 seconds to remove all the redundant rules from a packet classifier that has up to 6000 rules.

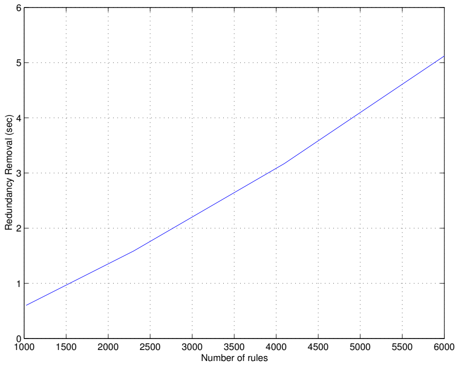


Figure 12. Average processing time for removing all (both upward and downward) redundant rules vs. Total number of rules in a packet classifier

7. Concluding Remarks

We make three major contributions in this paper. First, we propose to remove all redundant rules from a packet classifier before a packet classification algorithm starts building data structures from the rules. By this preprocessing procedure of removing redundant rules, both space and time for a packet classification algorithm are reduced. Second, we give a necessary and sufficient condition for identifying all redundant rules, based on which we categorize redundant rules into upward redundant rules and downward redundant rules. Third, we present two efficient graph based algorithms for detecting and removing these two types of redundant rules. The experimental results shows that in a few seconds our algorithms can remove all redundant rules from a packet classifier with thousands of rules. We believe that our redundancy removal algorithms will be a valuable preprocessing procedure for packet classification algorithms.

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

References

[1] N. Benameur, S. B. Fredj, S. Oueslati, and J. Roberts. Quality of service and flow level admission control in the internet. *Computer Networks*, 40:57–71, 2002.

[2] C. Dovrolis, D. Stiliadis, and P. Ramanathan. Proportional differentiated services: Delay differentiation and packet scheduling. In *Proc. of ACM SIGCOMM*, pages 109–120, 1999.

[3] 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)*, March 2004.

[4] P. Gupta. *Algorithms for Routing Lookups and Packet Classification.* PhD thesis, Stanford University, 2000.

[5] P. Gupta and N. McKeown. Packet classification on multiple fields. In *Proc. of ACM SIGCOMM*, pages 147–160, 1999.

[6] Java. http://java.sun.com/. November 2003.

[7] M. Kounavis, A. Kumar, H. Vin, R. Yavatkar, and A. Campbell. Directions in packet classification for network processors. *Network Processors Design: Issues and Practices*, 2, 2003.

[8] S. Merugu, S. Bhattacharjee, E. W. Zegura, and K. L. Calvert. Bowman: A node OS for active networks. In *Proc. of IEEE INFOCOM*, pages 1127–1136, 2000.

[9] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek. The click modular router. In *Symposium on Operating Systems Principles*, pages 217–231, 1999.

[10] M. H. Overmars and A. F. van der Stappen. Range searching and point location among fat objects. *Journal of Algorithms*, 21(3):629–656.

[11] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet classification using multidimensional cutting. In *Proc. of ACM SIGCOMM*, 2003.

[12] E. Spitznagel, D. Taylor, and J. Turner. Packet classification using extended tcams. In *Proc. of IEEE International Conference on Network Protocols (ICNP)*, November 2003.

[13] T. Y. C. Woo. A modular approach to packet classification: Algorithms and results. In *Proc. of IEEE INFOCOM*, pages 1213–1222, 2000.