PAPER Special Section on Foundations of Computer Science —Frontiers of Theory of Computation and Algorithm—

Simulated Annealing Method for Relaxed Optimal Rule Ordering

Takashi HARADA[†a)], **Ken TANAKA**[††b)], ***and*** **Kenji MIKAWA**[†††c)], ***Members***

SUMMARY Recent years have witnessed a rapid increase in cyber-attacks through unauthorized accesses and DDoS attacks. Since packet classification is a fundamental technique to prevent such illegal communications, it has gained considerable attention. Packet classification is achieved with a linear search on a classification rule list that represents the packet classification policy. As such, a large number of rules can result in serious communication latency. To decrease this latency, the problem is formalized as optimal rule ordering (ORO). In most cases, this problem aims to find the order of rules that minimizes latency while satisfying the dependency relation of the rules, where rules r_i and r_j are dependent if there is a packet that matches both r_i and r_j and their actions applied to packets are different. However, there is a case in which although the ordering violates the dependency relation, the ordering satisfies the packet classification policy. Since such an ordering can decrease the latency compared to an ordering under the constraint of the dependency relation, we have introduced a new model, called relaxed optimal rule ordering (RORO). In general, it is difficult to determine whether an ordering satisfies the classification policy, even when it violates the dependency relation, because this problem contains unsatisfiability. However, using a zero-suppressed binary decision diagram (ZDD), we can determine it in a reasonable amount of time. In this paper, we present a simulated annealing method for RORO which interchanges rules by determining whether rules r_i and r_j can be interchanged in terms of policy violation using the ZDD. The experimental results show that our method decreases latency more than other heuristics.

key words: *packet classification, relaxed optimal rule ordering,* ***NP****-hard, simulated annealing, zero-suppressed binary decision diagram*

1. Introduction

In recent years, cyber-attacks through unauthorized accesses and DDoS attacks have rapidly increased. Packet classification is a fundamental technique to prevent such illegal communications. Consequently, much research has been devoted to packet classification. Packet classifiers determine the behavior of incoming packets through a comparison with the packet classification policy. A classification policy is generally represented as a list of classification rules. Packet classification is achieved by comparisons until a match is found. The processing latency of packet classification is proportional to the number of rules, however. By placing rules that match many packets to an upper position, we can reduce the classification latency. Thus, an optimization problem, called *optimal rule ordering* (ORO), aims to find the order of rules that minimizes the classification latency. Give the reduction algorithm from the job scheduling problem, this problem is known to be **NP**-hard [1], [2].

In general, it is difficult to determine in a reasonable amount of time whether ordering a σ that violates the dependency relation satisfies the classification policy, because this problem contains a determination of unsatisfiability. However, in σ, using zero-suppressed binary decision diagrams (ZDDs) [3], [4] according to each set of packets evaluated by a rule in the rule list, we can determine it in realistic time. To determine whether an ordering that is modified locally satisfies the policy with ZDDs is not especially difficult. Moreover, these ZDDs can be used to calculate the weights of rules. Thus, in this paper, we present a simulated annealing method [5], [6] for relaxed optimal rule ordering (RORO) that interchanges dependent rules without any policy violation by determining with a ZDD whether interchanging them causes a policy violation.

The remainder of this article is organized as follows. Section 2 formulates RORO and introduces some terminology. Since our method uses a ZDD, we outline the ZDD in Sect. 3. We present our rule reordering method based on simulated annealing in Sect. 4. In Sect. 5, the efficiency of our methods is evaluated experimentally. The experimental results show that our method decreases the latency compared to other heuristics. Finally, Sect. 6 summarizes this paper and discusses tasks for future work.

2. Packet Classification

Packet classification on network devices is modeled as shown in Fig. 1. A packet p is a bit string of length w, i.e., $p \in \{0,1\}^l$. Each rule consists of a rule number $i \in \mathbb{N}$, a condition string on $\{0,1,*\}^l$, and an evaluation type $\{A_1, A_2, \ldots, A_m\}$, where l is the length of a condition. Further, $*$ is a so-called *don't care term* denoting that any bit can be matched, and m is the number of evaluation types. A rule list consists of n rules. Since an actual rule ordinarily contains *Permit* or *Deny* as the evaluation type, in this paper, the set of evaluation types consists only of Permit (P) and Deny (D). Here, P and D denote whether the device accepts or denies incoming packets, respectively. A rule is defined as shown in (1). An example of a rule list is provided in

Manuscript received April 25, 2019.
Manuscript revised September 26, 2019.
Manuscript publicized December 20, 2019.

[†]The author is with School of Information, Kochi University of Technology, Kami-shi, 782–8502 Japan.
[††]The author is with Department of Science, Kanagawa University, Hiratsuka-shi, 259–1293 Japan.
[†††]The author is with Center for Academic Information Service, Niigata University, Niigata-shi, 950–2181 Japan.
a) E-mail: harada.takashi@kochi-tech.ac.jp
b) E-mail: ktanaka@info.kanagawa-u.ac.jp
c) E-mail: mikawa@cais.niigata-u.ac.jp
DOI: 10.1587/transinf.2019FCP0006

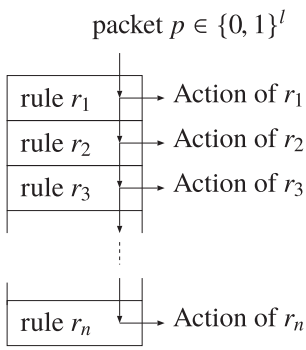

Fig. 1 Packet classification model.

Table 1 A rule list.

Filter $\mathcal{R}$	$\lvert E(\mathcal{R}, i)\rvert_{\mathcal{U}}$
$r_1^P = *0*1$	4
$r_2^P = 0000$	1
$r_3^P = 0*00$	1
$r_4^D = 0*1*$	3
$r_5^P = *1*1$	3
$r_6^D = ****$	4
$L(\mathcal{R}, \mathcal{U}) = 56$	

Table 2 Reordering according to σ.

Filter $\mathcal{R}_\sigma$	$\lvert E(\mathcal{R}_\sigma, i)\rvert_{\mathcal{U}}$
$r_1^P = *0*1$	4
$r_4^D = 0*1*$	3
$r_5^P = *1*1$	3
$r_3^P = 0*00$	2
$r_2^P = 0000$	0
$r_6^D = ****$	4
$L(\mathcal{R}_\sigma, \mathcal{U}) = 47$	

Table 1.

Definition 1: (rule form)

$$r_i^e = b_1 b_2 \cdots b_l, \;\; b_k \in \{0, 1, *\}, \;\; e \in \{P, D\} \tag{1}$$

Let an ordering of n rules be a bijective function $\sigma : [n] \to [n]$, where $[n] = \{1, 2, \ldots, n\}$. For instance,

$$\sigma = (1\ 5\ 4\ 2\ 3\ 6) \tag{2}$$

denotes $1 \to 1, 2 \to 5, 3 \to 4, 4 \to 2, 5 \to 3$, and $6 \to 6$. In this case, $\sigma(3) = 4$ means that the rule in the third position moves to the fourth position, and $\sigma^{-1}(4) = 3$ means that the rule in fourth position was moved from the third position. Informally, the domain and codomain of the function σ are a set of rule numbers and a set of positions for rules. Let $\mathcal{R}$ be a rule list and σ be an ordering. $\mathcal{R}_\sigma$ denotes the rule list reordered by σ. With the above ordering σ, the rule list

$$\mathcal{R} = [r_1^{e_1}, r_2^{e_2}, r_3^{e_3}, r_4^{e_4}, r_5^{e_5}, r_6^{e_6}]$$

is reordered as follows:

$$\mathcal{R}_\sigma = [r_1^{e_1}, r_4^{e_4}, r_5^{e_5}, r_3^{e_3}, r_2^{e_2}, r_6^{e_6}]$$

We use $\mathcal{R}(p)$ to denote an evaluation type for p as the classification result. For example, given the rule list $\mathcal{R}$ in Table 1, $\mathcal{R}(0011) = P$. The rule list in Table 1 denotes the function $f : \{0,1\}^4 \to \{P, D\}$ given in (3).

$$\begin{array}{llll} 0000 \mapsto P, & 0001 \mapsto P, & 0010 \mapsto D, & 0011 \mapsto P, \\ 0100 \mapsto P, & 0101 \mapsto P, & 0110 \mapsto D, & 0111 \mapsto D, \\ 1000 \mapsto D, & 1001 \mapsto P, & 1010 \mapsto D, & 1011 \mapsto P, \\ 1100 \mapsto D, & 1101 \mapsto P, & 1110 \mapsto D, & 1111 \mapsto P \end{array} \tag{3}$$

If there exists a packet p such that $\mathcal{R}(p) \neq \mathcal{R}_\sigma(p)$, we say that order σ violates the policy or that a policy violation occurs. For instance, order

$$\beta = (1\ 2\ 3\ 5\ 4\ 6)$$

violates the policy represented by the rule list in Table 1, because $D = \mathcal{R}(0111) \neq \mathcal{R}_\beta(0111) = P$.

Let $id : [n] \to [n]$ be the identity ordering, i.e., $id(i) = i$ for all $i \in [n]$. $\mathcal{R}_{id}$ means that the rule list is not reordered and $\mathcal{R}_{id}$ equals $\mathcal{R}$. In the following, id is omitted from a rule list inscription when the order of the rule list is id.

Let $M(r_i)$ denote a set of packets that can match rule r_i^e. That is, $M(r_i)$ is a set of binary sequences generated by changing each '$*$' on the condition of r_i^e to 0 or 1. For example, for r_4^D (Table 1),

$$M(r_4) = \{\ 0010, 0011, 0110, 0111\ \}.$$

Given a rule list $\mathcal{R}$ and an ordering σ, a set of packets evaluated by rule r_i^e is defined. This set is denoted as $E(\mathcal{R}_\sigma, i)$. As with $M(r_i)$, e is omitted. For example, given the rule list in Table 1, the set of packets evaluated by rule r_4^D is expressed as

$$E(\mathcal{R}, 4) = \{\ 0010, 0110, 0111\ \}.$$

Note that $E(\mathcal{R}, 4)$ is different from $M(r_4)$. As the packet 0011 is evaluated by rule r_1^P, 0011 is not in $E(\mathcal{R}, 4)$.

Let $\mathcal{F} : \{0,1\}^l \to \mathbb{N}$ be a packet arrival frequency distribution, and let $\lvert\mathcal{P}\rvert_{\mathcal{F}}$ denote $\sum_{p \in \mathcal{P}} \mathcal{F}(p)$. For example, for $\mathcal{P} = \{\ 0010, 0101, 1001\ \}$ and $\mathcal{F}$ in (4),

$$\begin{aligned} \lvert\mathcal{P}\rvert_{\mathcal{F}} &= \lvert\{\ 0010, 0101, 1001\ \}\rvert_{\mathcal{F}} \\ &= \mathcal{F}(0010) + \mathcal{F}(0101) + \mathcal{F}(1001) = 8. \end{aligned}$$

$$\begin{array}{llll} 0000 \mapsto 20, & 0001 \mapsto 0, & 0010 \mapsto 7, & 0011 \mapsto 0, \\ 0100 \mapsto 10, & 0101 \mapsto 0, & 0110 \mapsto 0, & 0111 \mapsto 0, \\ 1000 \mapsto 7, & 1001 \mapsto 1, & 1010 \mapsto 13, & 1011 \mapsto 0, \\ 1100 \mapsto 0, & 1101 \mapsto 0, & 1110 \mapsto 0, & 1111 \mapsto 0. \end{array} \tag{4}$$

Given a packet arrival distribution $\mathcal{F}$, a rule list $\mathcal{R}$, and an order of rules σ, the number of packets evaluated by r_i under $\mathcal{F}$ can be defined. We denote this number as $\lvert E(\mathcal{R}_\sigma, i)\rvert_{\mathcal{F}}$ and call it the *weight of* r_i. For example, under a uniform distribution $\mathcal{U} : \mathcal{P} \to \{1\}$, the number of evaluated packets r_3 in Table 1 is $\lvert E(\mathcal{R}_\sigma, 3)\rvert_{\mathcal{U}} = 1$. Considering that the

comparison of a packet with a rule has latency 1, under the order of rules σ and the packet arrival distribution $\mathcal{F}$, the classification latency $L(\mathcal{R}_\sigma, \mathcal{F})$ of rule list $\mathcal{R}$ is defined as follows:

Definition 2: (Classification latency)

$$L(\mathcal{R}_\sigma, \mathcal{F}) = \sum_{i=1}^{n-1} i|E(\mathcal{R}_\sigma, \sigma^{-1}(i))|_{\mathcal{F}} + (n-1)|E(\mathcal{R}_\sigma, \sigma^{-1}(n))|_{\mathcal{F}}. \tag{5}$$

In other words, latency can be expressed as

$$L(\mathcal{R}_\sigma, \mathcal{F}) = \sum_{i=1}^{n} |E(\mathcal{R}_\sigma, i)|_{\mathcal{F}} \cdot \sigma(i) - |E(\mathcal{R}_\sigma, \sigma^{-1}(n))|_{\mathcal{F}}$$

in terms of the rule number. As a packet is not compared with the last rule $r_{\sigma^{-1}(n)}$, the second term is necessary. For example, the classification latency for the rule list in Table 1 with a uniform distribution $\mathcal{U}$ is expressed as

$$L(\mathcal{R}, \mathcal{U}) = 1{\cdot}4 + 2{\cdot}1 + 3{\cdot}1 + 4{\cdot}3 + 5{\cdot}3 + 5{\cdot}4 = 56.$$

By reordering the rules in Table 1 according to σ while maintaining the classification policy denoted by f, the latency decreases from 60 to

$$L(\mathcal{R}_\sigma, \mathcal{U}) = 1{\cdot}4 + 2{\cdot}3 + 3{\cdot}3 + 4{\cdot}2 + 5{\cdot}0 + 5{\cdot}4 = 47.$$

As described above, by reordering the rules, the classification latency of a rule list can be decreased. In addition, for each rule r_i, reordering the rules may vary the number of packets evaluated by r_i. Therefore, the optimal order of rules actually varies according to the packet arrival distribution. To clarify the optimal order for rules, we now define RORO with a given packet arrival distribution.

Definition 3: (RORO)

Input: Rule list $\mathcal{R}$ and packet arrival distribution $\mathcal{F}$
Output: Order of rules σ that minimizes $L(\mathcal{R}_\sigma, \mathcal{F})$ s.t. $\forall p \in \mathcal{P},\ \mathcal{R}(p) = \mathcal{R}_\sigma(p)$

In the above definition, $\forall p \in \mathcal{P},\ \mathcal{R}(p) = \mathcal{R}_\sigma(p)$ means that order σ does not violate the policy represented by rule list $\mathcal{R}$. That is, order σ is a feasible solution.

In order to explain the difference of RORO and conventional ORO whose constraint is based on overlap or dependency relation on rules, we define the *overlap* and *dependency* of the rules.

Definition 4: (Overlap of rules) If there is a packet that matches both r_i and r_j, r_i and r_j are said to overlap.

For example, rules r_4^D and r_6^D in Table 1 overlap, because packet 0010 exists and matches both r_4^D and r_6^D.

Definition 5: (Dependency of rules) If r_i and r_j overlap and the evaluation type of r_i is different from that of r_j, we say that r_i and r_j are dependent.

As an instance, rules r_4^D and r_5^P are dependent, because they overlap, owing to 0111, and their evaluation types are different. We say that ordering σ satisfies the overlap relation if, for any overlapping rules r_i and r_j $i < j$, r_i is placed ahead of r_j in σ. We say that ordering σ satisfies the dependency relation if, for any dependent rules r_i and r_j $(i < j)$, r_i is placed ahead of r_j in σ. Most researchers define a constraint for ORO such that an ordering must satisfy the overlap relation [2], [7]–[12].

Although an ordering violates the overlap relation, it can satisfy the classification policy if it satisfies the dependency relation. Therefore, Tanaka et al. defined the constraint of ORO such that an ordering must satisfy the dependency relation [13], [14]. For example, although rules r_2^P and r_3^P in Table 1 overlap, rule list $[r_1^P, r_3^P, r_2^P, r_4^D, r_5^P, r_6^D]$ that violates the overlap relation satisfies the policy. This is because packet 0000, which makes r_2^P and r_3^P overlap, turns out to receive evaluation type P with r_3^P.

Further, there is a case, in which an ordering satisfies the classification policy while violating the dependency relation. For instance, although order $\pi = (1\ 6\ 2\ 3\ 4\ 5)$ violates the dependency relation, rule list $\mathcal{R}_\pi = [r_1^P, r_3^P, r_4^D, r_5^P, r_6^D, r_2^P]$ satisfies the policy. Placing r_6^D ahead of r_2^P violates the dependency relation with respect to packet 0000. However, packet 0000 is properly evaluated by r_3^P before r_6^D. Thus, there is no policy violation. Hence, Misherghi et al. defined ORO in terms of policy violation instead of the overlap or dependency relation [15], and we defined RORO.

Existing heuristics for ORO based on the overlap or dependency relations [2], [7]–[14] do not interchange dependent rules r_i and r_j. However, if the weight of r_j is large and if, dependent rules r_i and r_j can be interchanged without a policy violation, interchanging them considerably decreases the latency. Therefore, we propose a reordering algorithm that interchanges dependent rules in the above situation in Sect. 4.

3. ZDD

Since our reordering method uses ZDD, this section outlines the ZDD [3], [4].

A combination of l items can be represented by a l-bit vector $(b_1, b_2, \ldots, b_l)$, where each b_k expresses whether or not the combination contains the item. A set of combinations can be represented by a set of l-bit vectors. A set of evaluated packets can also be regarded as a set of combinations.

The ZDD data structure was proposed by Minato to manipulate a set of combinations efficiently [3]. A ZDD is obtained by applying reduction rules to a binary decision tree representing a set of combinations. Deleting a redundant node and sharing an identical node are illustrated in Figs. 4 and 5, respectively. Figures 2 and 3 represent the same set of combinations $\{001, 111\}$. Circles denote non-terminal nodes and boxes indicate terminal nodes. A numeral i associated with a non-terminal node represents a Boolean variable of item i. Non-terminal nodes have edges with values of 1 and 0. The 1 and 0 edges of node i ex-

press whether or not this node contains item i. In ZDDs, the variables are ordered. On the path from the root node to a terminal node, indicated by a bold arrow, a skipped variable i indicates that the combination does not contain item i.

Figure 6 shows the ZDDs $M(r_i)$s and $E(\mathcal{R}, i)$ according to rule list $\mathcal{R}$ in Table 1.

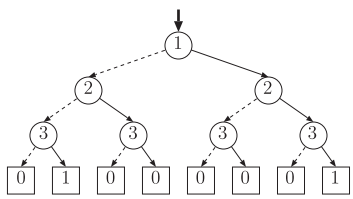

Fig. 2 Binary decision tree.

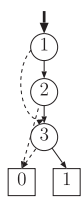

Fig. 3 ZDD.

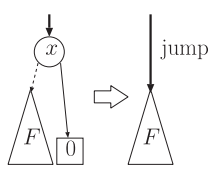

Fig. 4 Node deletion.

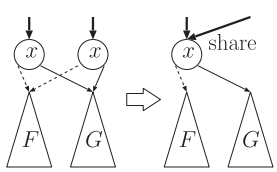

Fig. 5 Node sharing.

4. Simulated Annealing Method for RORO

In this section, we present a method for determining with ZDDs whether interchanging dependent rules satisfies the policy. We also propose simulated annealing with this determination method.

4.1 Determining Whether Interchanging Dependent Rules Satisfies the Policy

Under ordering σ, if adjacent rules r_i and r_j are dependent and the weight of the posterior rule r_j is large, interchanging them can reduce the classification latency. Recall that $M(r_i)$ is the set of packets that match rule r_i regardless of other rules in the rule list, and that $E(\mathcal{R}_\sigma, i)$ is the set of packets that are evaluated by rule r_i under ordering σ. In order to perform the interchange described above under the ordering σ, the sets $E(\mathcal{R}_\sigma, i)$ and $M(r_j)$ are important. This is because, although r_i and r_j are dependent as $M(r_i) \cap M(r_j) \neq \emptyset$ and their evaluation types are different, interchanging dependent rules r_i^e and r_j^f does not violate policy when $E(\mathcal{R}_\sigma, i) \cap M(r_j) = \emptyset$.

Consider the rule list $\mathcal{R}$ in Table 1 and ordering σ (2). The rules r_2^P and r_6^D overlap, owing to packet 0000, and they are dependent since their evaluation types are different. Here, we need to determine whether we can interchange these dependent rules. Under ordering σ, the set of packets evaluated by rule r_2^P is $E(\mathcal{R}_\sigma, 2) = \emptyset$, and the set of packets that matches r_6^D is $M(r_6) = \{0000, 0001, \ldots, 1111\}$. Since $E(\mathcal{R}_\sigma, 2) \cap M(r_6) = \emptyset$, interchanging r_2^P and r_6^D does not produce any packets that receive an evaluation type that is different from the policy. Therefore, under ordering σ, we can interchange r_2^P and r_6^D.

In contrast to the above rules, under the same rules and ordering, we can not interchange the dependent rules r_4^D and r_5^P. Because $E(\mathcal{R}_\sigma, 4) \cap M(r_5) = \{0111\}$, interchanging r_4^D

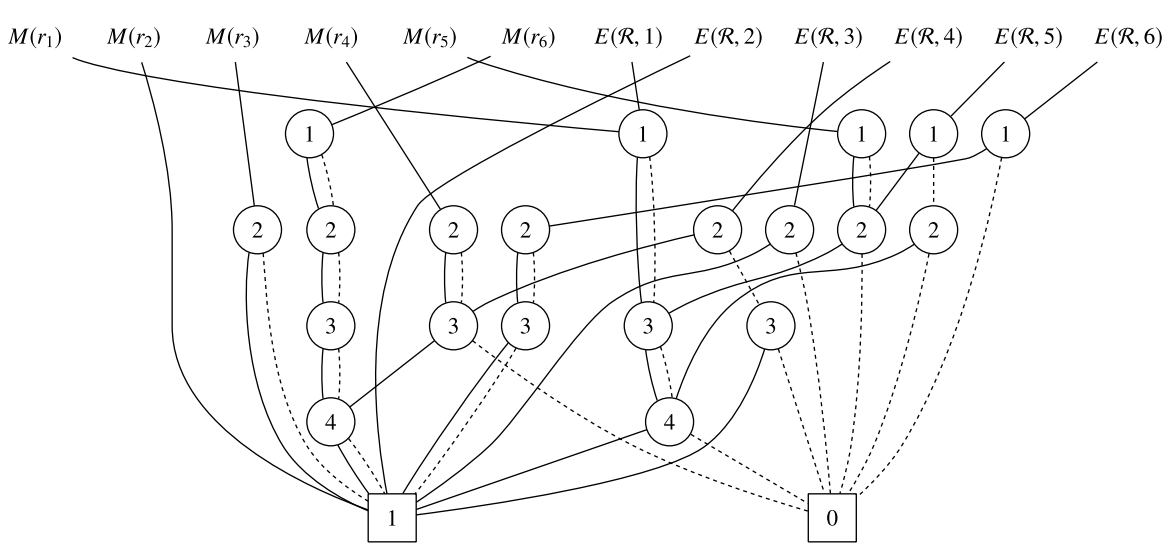

Fig. 6 ZDDs according to the rule list in Table 1.

Algorithm 1: Determination of whether placing r_j before r_i satisfies the policy

input : Rule list $\mathcal{R}$, rule ordering σ, and rule numbers i, j, s.t. $\sigma(j) - \sigma(i) = 1$
output: *true* if interchanging adjacent rules yet satisfies the policy, *false* otherwise
set Z_i to the ZDD for $E(\mathcal{R}_\sigma, i)$;
set Z_j to the ZDD for $M(r_j)$;
if $Z_i \cap Z_j$ *points to* 0 *terminal node* **then return** *true* ;
else return *false* ;

Algorithm 2: Determination of whether interchanging rules satisfies the policy

input : Rule list $\mathcal{R}$, rule ordering σ, and rule numbers i, j, s.t. $\sigma(i) < \sigma(j)$
output: *true* if interchanging rules yet satisfies the policy, *false* otherwise
// comparing $E(\mathcal{R}_\sigma, i), E(\mathcal{R}_\sigma, \sigma^{-1}(\sigma(i)+1)), \ldots,$ $E(\mathcal{R}_\sigma, \sigma^{-1}(\sigma(j)-1))$ with $M(r_j)$;
1 $k \leftarrow \sigma(i)$;
2 **while** $k < \sigma(j)$ **do**
3 flag $\leftarrow$ Algorithm 1$(\mathcal{R}, \sigma, \sigma^{-1}(k), j)$;
if $\neg$ *flag* **then return** *false*;
4 $k \leftarrow k + 1$;
end
// comparing $M(r_{\sigma^{-1}(\sigma(j)-1)}), M(r_{\sigma^{-1}(\sigma(j)-2)}), \ldots, M(r_{\sigma^{-1}(\sigma(i)+1)})$ with $E(\mathcal{R}_\sigma, i)$;
5 $k \leftarrow \sigma(j) - 1$;
6 **while** $\sigma(i) < k$ **do**
7 flag $\leftarrow$ Algorithm 1$(\mathcal{R}, \sigma, i, \sigma^{-1}(k))$;
if $\neg$ *flag* **then return** *false*;
8 $k \leftarrow k - 1$;
end
9 **return** *true*;

and r_5^P gives the evaluation type D for packet 0111 even though P must be applied to 0111. That is, a policy violation occurs.

We show the algorithm that determines whether we can interchange adjacent rules r_i and r_j ($i < j$) without a policy violation under ordering σ in Algorithm 1, where we suppose that the ZDD for $E(\mathcal{R}, i)$ of each r_i in the rule list has already been constructed. Algorithm 1 takes a rule list $\mathcal{R}$, an ordering σ, and rule numbers i and j such that $\sigma(i) + 1 = \sigma(j)$ under σ as inputs. It returns *true* if we can interchange those rules, and returns *false*, otherwise.

Next, Algorithm 2 determines whether rules r_i and r_j can be interchanged under ordering σ. Note that the algorithm does not require that r_i and r_j are adjacent in ordering σ. Algorithm 2 determines whether rule r_j can be interchanged with the rules from $\sigma(i)$ to $\sigma(j) - 1$ under ordering σ in lines 2–4, and determines whether rule r_i can be interchanged of rules at $\sigma(j) - 1$ to $\sigma(i) + 1$ under σ in lines 6–8.

When r_i and r_j are interchanged in Algorithm 2, we should update evaluation packets sets ZDDs for rules r_k in $\sigma(i) \le k \le \sigma(j)$. We update $E(\mathcal{R}_\sigma, \sigma^{-1}(k))$ as follows:

$$E(\mathcal{R}_\sigma, \sigma^{-1}(k)) \leftarrow E(\mathcal{R}_\sigma, \sigma^{-1}(k)) \cup Z_{ik} \setminus Z_{jk}$$

Algorithm 3: Simulated Annealing

input : Rule list $\mathcal{R}$,
Frequency distribution $\mathcal{F}$,
Inner loop parameter *loop*,
Temperature factor *temp*,
Freezing parameter *freeze*
output: Rule ordering σ
$\sigma \leftarrow$ order obtained by the method [13] as an initial order;
$o \leftarrow$ searching order for $|N(\sigma)|$;
$t \leftarrow 200$;
$j \leftarrow 0$;
while $j <$ *freeze* $\cdot |N(\sigma)|$ **do**
$i \leftarrow 0$;
while $i <$ *loop* $\cdot |N(\sigma)|$ **do**
get $\sigma' \in N(\sigma)$ according to o ;
$\Delta = h(\sigma, \sigma')$;
if $\Delta \le 0$ **then** $\sigma \leftarrow \sigma'$, $j \leftarrow 0$;
else
choose a random value $q \in [0, 1]$;
if $q < e^{-\Delta/t}$ **then** $\sigma \leftarrow \sigma'$;
$j \leftarrow j + 1$;
end
$i \leftarrow i + 1$;
end
$t \leftarrow t \cdot$ *temp*
end

where, $Z_{ik} = E(\mathcal{R}_\sigma, i) \cap M(r_k)$ and $Z_{jk} = E(\mathcal{R}_\sigma, \sigma^{-1}(k)) \cap M(r_j)$.

In [15], Misherghi et al. use a binary decision diagram (BDD) [16]–[18] to partition the packet space. Our determination algorithm can likewise be based on BDD instead of ZDD. However, since practical rules consist of few don't cares and $E(\mathcal{R}_\sigma, i)$ is so sparse, ZDD can more efficiently manage $E(\mathcal{R}_\sigma, i)$ for each rule r_i in the rule list in terms of the number of nodes required. Thus, we utilize ZDD instead of BDD.

4.2 Simulated Annealing by Exchanging Dependent Rules without Any Policy Violation

In this subsection, we present a simulated annealing method based on the above determination algorithm of policy equivalence. Since it is important to place a heavy rule in the upper position to reduce the latency, we define the neighbor of a solution σ used in the simulated annealing method as follows:

$$N(\sigma) = \left\{\tau_{ij} \circ \sigma \mid \mathcal{R}_{\tau_{ij} \circ \sigma} \text{ holds policy}\right\} \tag{6}$$

where, τ_{ij} is the transposition interchanging r_i and r_j. Note that the solution in the neighbor is only a feasible solution and that, using ZDDs, the solution is determined exhaustively by whether it is feasible.

Our method makes the random order $o : [|N(\sigma)|] \rightarrow [|N(\sigma)|]$ and search an improved solution in $N(\sigma)$ in this order.

The evaluation function h of solutions σ and $\sigma' \in N(\sigma)$ is defined as follows:

$$h(\sigma, \sigma') = w(i) - w(\sigma(i) - a), \tag{7}$$

where $\sigma' = (\sigma(i) - a \;\; \sigma(i)) \circ \sigma$, and $w(i)$ is the weight of rule r_i. Although we should consider the variation of the weights of rules when interchanging overlapping rules as claimed in [19], considering it takes a lot of resource and, we thus ignore it.

We show a simulated annealing method based on the neighbor (6) and the evaluation function (7) in Algorithm 3. Algorithm 3 takes rule list $\mathcal{R}$, packet arrival distribution $\mathcal{F}$, loop parameter *loop*, cooling temperature *freeze*, and decrease ratio *temp* as inputs and returns the ordering of rules.

Since our method defines the neighbor as Eq. (6) and ignores transitions of light rules, the method does not exhaustively search for the solution space. Thus, if we generate an initial solution at random, we cannot obtain a good solution. Therefore, our method uses the solution obtained by applying the reordering method of Tanaka et al. [13] as the initial solution on line 1.

5. Experiments

We demonstrated the efficiency of the proposed method through experiments. The proposed method was implemented in C++ under CentOS 7.3 on an Intel Core i7 3.2 GHz with 8 GB main memory. We used the CUDD ZDD package [20]. For the comparison, we implemented a simple rule sorting algorithm (SR) [7], the method by Tanaka et al. [13], sub-graph merging (SGM) [10], and the method by Hikage et al. [11] in Java. We generated rules and headers with the standard packet benchmark tool for packet classification algorithms, namely, ClassBench [21]. For the experiments on RORO, we added an evaluation type P or D to each rule in rule lists generated by ClassBench with a probability of 1/2. A packet header generated by ClassBench consists of source/destination addresses, source/destination port numbers, and the protocol number. Since the lengths of these components are 32, 32, 16, 16, and 8 bits, respectively, the total length of the packet header and the condition of the rule is 104 bits. The number of headers generated according to each rule list was approximately 100,000.

Using the generated rule lists and headers, we measured the time for reordering rules and the latency of a rule list for every algorithm. The units of measurement is seconds. The means of 10 trials for RORO are shown in Figs. 7 and 8. Note that we plot the reordering times on a logarithmic scale in Fig. 8. In addition, the construction time of ZDDs $M(r_i)$s and $E(\mathcal{R}, i)$s is under a second. With our method, therefore, the ZDDs can be constructed in a practical amount of time.

As shown in Fig. 7, the proposed method decreases the latency by about 11% compared to Tanaka et al. [13], SGM [10], and Hikage et al. [12]. This 11% reduction of classification latency is large improvement. Moreover, as the number of rules becomes large, the proposed method decreases the latency compared to the other heuristics.

Figure 8 shows that the proposed method can terminate

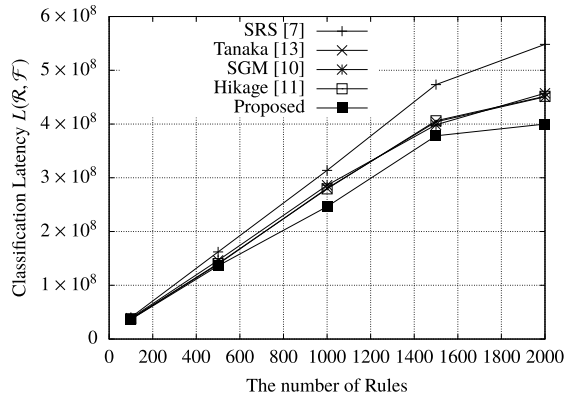

Fig. 7 Latency for SRS [7], Tanaka [13], SGM [10], Hikage [11], and the proposed method.

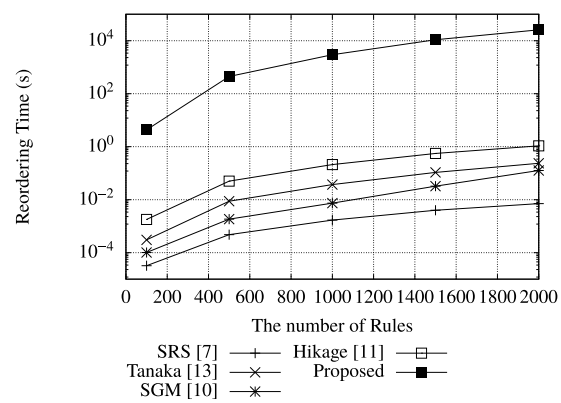

Fig. 8 Reordering time (s) for the proposed method.

for approximately 2,000 rules in half a day. Thus, it is fast, owing to the use of simulated annealing. Furthermore, as shown in Fig. 8, the time for reordering rules is not exponential to the number of rules.

6. Conclusion

In this paper, we presented a simulated annealing method for RORO based on policy violation rather than the overlap or dependency relations on rules. The proposed method focuses on an ordering that satisfies the policy yet does not satisfy the dependency relation using ZDD. Such an ordering can not be obtained when we focus exclusively on the dependency relation, and its latency is often less than that of orderings in the dependency relation. Experimental results showed that the proposed method decreased the latency compared to other heuristics.

By pruning redundant searches in the neighbor, reducing the reordering time of the proposed method is an important task.

The proposed method determines whether an order satisfies the policy using ZDDs. Since the worst-case space complexity of ZDD is exponential and length of rules, there is a case where a ZDD cannot be made according to each rule in the rule list. Thus, a reordering algorithm should be

developed that determines whether an ordering satisfies the policy yet does not satisfy the dependency relation. We will pursue this in future work.

Acknowledgments

This work was partially supported by JSPS KAKENHI Grant Numbers 19K11953 and 19K11959.

References

[1] E.L. Lawler, "Sequencing jobs to minimize total weighted completion time subject to precedence constraints," Algorithmic Aspects of Combinatorics, Annals of Discrete Mathematics, vol.2, pp.75–90, 1978.
[2] H. Hamed and E. Al-Shaer, "Dynamic rule-ordering optimization for high-speed firewall filtering," Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, ASIACCS '06, New York, NY, USA, pp.332–342, ACM, 2006.
[3] S. Minato, "Zero-suppressed bdds for set manipulation in combinatorial problems," 30th ACM/IEEE Design Automation Conference, pp.272–277, June 1993.
[4] S. Minato, "Zero-suppressed bdds and their applications," International Journal on Software Tools for Technology Transfer, vol.3, no.2, pp.156–170, 2001.
[5] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon, "Optimization by simulated annealing: An experimental evaluation; part i, graph partitioning," Oper. Res., vol.37, no.6, pp.865–892, Oct. 1989.
[6] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon, "Optimization by simulated annealing: An experimental evaluation; part ii, graph coloring and number partitioning," Oper. Res., vol.39, no.3, pp.378–406, May 1991.
[7] E.W. Fulp, "Optimization of network firewall policies using directed acyclic graphs," Proceedings IEEE Internet Management Conf, extended abstract, 2005.
[8] H. Hazem, A. El-Atawy, and E. Al-Shaer, "Adaptive statistical optimization techniques for firewall packet filtering," INFOCOM, pp.1–12, IEEE, April 2006.
[9] E.-S.M. El-Alfy and S.Z. Selim, "On optimal firewall rule ordering," 2007 IEEE/ACS International Conference on Computer Systems and Applications, pp.819–824, May 2007.
[10] A. Tapdiya and E.W. Fulp, "Towards optimal firewall rule ordering utilizing directed acyclical graphs," Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on, pp.1–6, Aug. 2009.
[11] K. Hikage and T. Yamada, "Algorithm for minimizing overhead of firewall with maintenance of rule dependencies," Proceedings IEICE General Conference 2016, vol.2016, no.1, p.6, March 2016 (in Japanese).
[12] R. Mohan, A. Yazidi, B. Feng, and B.J. Oommen, "Dynamic ordering of firewall rules using a novel swapping window-based paradigm," Proceedings of the 6th International Conference on Communication and Network Security, ICCNS '16, New York, NY, USA, pp.11–20, ACM, 2016.
[13] K. Tanaka, K. Mikawa, and K. Takeyama, "Optimization of packet filter with maintenance of rule dependencies," IEICE Communications Express, vol.2, no.2, pp.80–85, Feb. 2013.
[14] K. Tanaka, K. Mikawa, and M. Hikin, "A heuristic algorithm for reconstructing a packet filter with dependent rules," IEICE Trans. Commun., vol.96, no.1, pp.155–162, Jan. 2013.
[15] G. Misherghi, L. Yuan, Z. Su, C.-N. Chuah, and H. Chen, "A general framework for benchmarking firewall optimization techniques," IEEE Transactions on Network and Service Management, vol.5, no.4, pp.227–238, Dec. 2008.
[16] S.B. Akers, "Binary decision diagrams," IEEE Transactions on Computers, vol.C-27, no.6, pp.509–516, June 1978.
[17] R.E. Bryant, "Graph-based algorithms for boolean function manipulation," IEEE Transactions on Computers, vol.C-35, no.8, pp.677–691, Aug. 1986.
[18] K.S. Brace, R.L. Rudell, and R.E. Bryant, "Efficient implementation of a bdd package," 27th ACM/IEEE Design Automation Conference, pp.40–45, June 1990.
[19] T. Harada, K. Tanaka, and K. Mikawa, "A heuristic algorithm for relaxed optimal rule ordering problem," 2018 2nd Cyber Security in Networking Conference (CSNet), pp.1–8, Oct. 2018.
[20] F. Somenzi, "Cudd package." http://vlsi.colorado.edu/~fabio/CUDD/cudd.pdf.
[21] D.E. Taylor and J.S. Turner, "Classbench: A packet classification benchmark," IEEE/ACM Trans. Netw., vol.15, no.3, pp.499–511, June 2007.

Takashi Harada received his B.S., M.S. and D.S. degrees from Kanagawa University in 2014, 2016 and 2019, respectively. He is currently a Research Associate in the school of Information at Kochi University of Technology.

Ken Tanaka received his B.E. degree from Waseda University in 1989 and M.E. degree from Tokyo Institute of Technology in 1992. He was an Assistant Professor at Niigata University, and is currently a Professor in the Faculty of Science at Kanagawa University. His interests include neural networks and network security technologies. He received his doctoral degree of Information Science from Waseda University. He is a member of IPSJ.

Kenji Mikawa received his B.E., M.E. and D.E. degrees in computer science from Ibaraki University in 1995, 1997 and 2001, respectively. He was an Assistant Professor at Ibaraki University and is now an Associate Professor at Niigata University. His interests include combinatorial algorithms and mathematical recreations. He is a member of LA symposium.