SwinTop: Optimizing Memory Efficiency of Packet Classification in Network Devices

Chang Chen
Department of Automation,
Tsinghua University,
Beijing, China
chenchang13@mails.tsinghua.edu.cn

Yang Xiang
Research Institute of Information Technology,
Tsinghua University,
Beijing, China
sharang@tsinghua.edu.cn,

Liangwei Cai
College of Information Engineering,
Shenzhen University,
Shenzhen, Guangdong, China
cailw@szu.edu.cn

Jun Li
Tsinghua National Lab for Information Science and Technology,
Tsinghua Universit,
Beijing, China
junl@tsinghua.edu.cn

***Abstract*—Packet classification is one of the key functionalities provided by network devices for QoS and network security purposes. Recently the rapid growth of classification ruleset size and ruleset complexity has caused memory performance woes when applying traditional packet classification algorithms. Inheriting the divide-and-conquer idea of pre-partitioning the original rules into several groups for significant reduction of memory overhead, this paper proposes Swin Top, a new ruleset partitioning approach based on swarm intelligent optimization algorithms, to seek for the global optimum grouping of rules. To enhance convergence accuracy and speed up the iterative process, Swin Top employs several novel ideas, such as the introduction of grouping penalty, the combination of PSO and GA, and a new memory usage estimation method. On the publicly available rulesets from Class Bench, SwinTop is shown to achieve 1 to 4 orders of magnitude lower memory consumption than simply applying a traditional packet classification algorithm without ruleset partitioning, and outperform the state-of-the-art partitioning algorithms EffiCuts and ParaSplit on all kinds of large-sized rulesets.**

Keywords - QoS; network security; packet classification; memory efficiency; ruleset partitioning; swarm intelligence optimization

I. INTRODUCTION

The rapid development of the Internet and the fast increase of network services have brought great challenges to deploying high-speed and quality-of-service (QoS) guaranteed networks. Besides packet forwarding, modern network devices need to provide more advanced services, such as access control, firewall, IDS (intrusion detection system) and VPN (virtual private network), etc. All these functionalities require the network devices to identify the packets being transmitted.

Packet classification, as a fundamental technique employed by network devices, is the process of classifying packets based on pre-defined rules. Each rule specifies a desired action (e.g., drop, forward) on a set of packets identified by some specific fields of the packet header (e.g., source IP, destination IP, source port, destination port, protocol type).

Although packet classification has been widely studied for years [1] [2], researchers are still motivated to seek novel packet classification solutions to keep pace with the emerging applications on the Internet. Recently it has been noticed that despite the optimization for throughput, the memory performance of software-based algorithms has become one of the major issues regarding the practical usage (or even the feasibility of implementation) of packet classification. The reasons are as follows in Table I:

TABLE I. MEMORY CONSUMPTION OF HYPERSPLIT

Ruleset	FW_1K	FW_10K	IPC_1K	IPC_10K
# rules	791	9311	938	9037
Memory consumption	3.6MB	1.01GB	1.5MB	66.8MB

Space inefficiency of traditional algorithms: Traditional packet classification algorithms have relatively low memory efficiency, and the memory requirements are in-deterministic with the ruleset size [3]. Specifically, the sizes of the data structures (e.g., decision trees) generated by these algorithms may inflate to multiple times their common sizes if the ruleset size scales up or the rules are extensively overlapped. As shown in Table I, the size of the decision tree generated by HyperSplit (the state-of-the-art decision tree based

978-1-4799-1984-0 /15/$31.00 ©2015 IEEE

algorithm) [4] for a real-life ruleset IPC_1K [5] (with 916 five-dimensional rules) is only 85*KB*, while that for ruleset FW_10K (with 9,311 five-dimensional rules) is up to 1.01*GB*.

Fast growth of real-life ruleset size: In order for fine-grained management, the ruleset size of a server at today's multi-tenant data centers can be up to 200K [6], causing memory performance woes to traditional packet classification schemes.

Memory limitation for running packet classification: The memory used for running packet classification is always limited, for which the concerns are twofold. First, the data structure size required for a complex ruleset may exceed the available memory size (e.g., the quota memory size for Xen hypervisor is less than 1GB). Second, the memory-consuming algorithms have to be implemented with mass-but-slow memories (e.g., SDRAMS), which undermines the classification speed.

TABLE II. AN EXAMPLE 2-DIMENSIONAL RULESET

Rule	Dimension X	Dimension Y	Action
R1	[4,5]	[0,7]	Drop
R2	[0,7]	[0,1]	Forward
R3	[4,7]	[4,7]	Forward
R4	[0,1]	[6,7]	Drop
R5	[0,3]	[0,7]	Drop
R6 (default)	[0,7]	[0,7]	Forward

Recently, several advanced solutions [3][7][8][9] have shown the superiority of ruleset partitioning in the improvement of memory performance. The basic idea is to trade classification performance for significant reduction in memory requirement, which can be achieved by partitioning the original ruleset into several sub-rulesets and build independent decision trees for the subsets using traditional packet classification algorithms. However, the existing solutions either scale poor as the dimension number grows, or bring uncertainty in eliminating rule replications and thus trap into local optimum.

Going beyond existing solutions, this paper presents a novel ruleset partitioning algorithm named *SwinTop* (SWarm INTelligence Optimization based Partitioning) that seeks for the global optimum grouping of rules. The main contributions of this paper include:

The ruleset partitioning problem is modeled as an integer programming problem with a huge and unsmooth search space, for which it is suitable to apply swarm intelligence algorithms. Moreover, inspired by the selective tree merging procedure of a previous work, the grouping penalty of rule pairs is defined to further provide guidance in the rule grouping process.

A novel intelligent ruleset partitioning algorithm is proposed for acquiring the global optimum grouping of rules in terms of memory consumption. The proposed algorithm is based on Particle Swarm Algorithm and Generic Algorithm, which are revised and combined in our approach in order to adapt to the ruleset partitioning problem.

A new method of estimating the memory consumption trends is proposed for significantly improving the time efficiency of the ruleset partitioning process.

The performance of SwinTop is evaluated on 18 rulesets of various sizes ranging from 100 to 50K rules. Experimental results show that on the large-sized rulesets, SwinTop achieves 1 to 4 orders of magnitude lower memory consumption than applying a traditional packet classification algorithm without ruleset partitioning. Compared with the state-of-the-art partitioning algorithms, SwinTop requires 30%~95% less memory than EffiCuts [7] and 20%~35% less memory than ParaSplit [9].

The rest of the paper is organized as follows: Section II introduces the background and relates work; Section III describes some preliminaries and Section IV presents the proposed algorithm in detail; Section V introduces the optimizations to the implementation of the algorithm; Section VI provides evaluation results; Section VII concludes the paper.

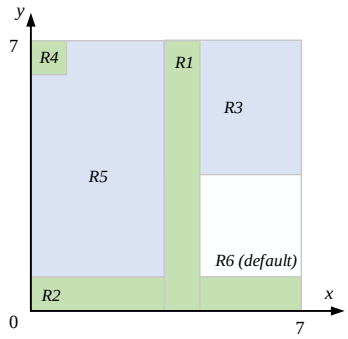


Figure 1. The geometrical model of the 2-D example ruleset

II. BACKGROUND

A. The Packet Classification Problem

The purpose of packet classification is to find the matching rule from a pre-defined ruleset for a packet. Each rule *R* contains *D* fields, and each field is a range match expression on a selected field of the packet header. Mathematically, packet classification can be viewed as a point location problem in computational geometry: In a *D*-dimensional search space *S*, a packet *P* is viewed as a *point* and a rule *R* is viewed as a *D*-dimensional *hyper-rectangle*. A packet *P* matches a rule *R* if the point represented by *P* locates inside the hyper-rectangle specified by *R*. If a packet hits more than one rules (some rules may be overlapped with each other), among them the highest-priority rule will be the final result. Table II shows an example ruleset with 6 two-dimensional rules, and Figure I shows the geometrical model of the ruleset.

The theoretical complexity bounds derived from computational geometry show that a packet classifier with N rules and D fields need either $\Theta(logN)$ time and $\Theta(N^D)$ space; or $\Theta(log_{D-1}N)$ time and $\Theta(N)$ space [10]. Thus in theory, even a ruleset with 1K five-dimensional rules can consume 1000*TB* memory in the worst case.

In practice, fortunately, the complexity can be significantly reduced by the well-designed packet classification algorithms for two reasons. First, despite the variety in statistical characteristics, the complexity of real-life rules is always far less than the theoretical worst case [4]. The common distribution patterns lying in the rules can be leveraged for lowering the size and depth of the decision trees. Second, given the ruleset, the classification data structure is generated during pre-processing (offline), and hence can be optimized in numbers of ways to achieve fast classification speed with rational memory usage.

B. Traditional Packet Classification Algorithms

Most well-known traditional packet classification algorithms are based on *search space decomposition*: the search space is decomposed into multiple sub-spaces, each of which is associated with a subset of rules (with rule replication); by recursively applying the decomposition, a decision tree is finally built for the run-time classification process.

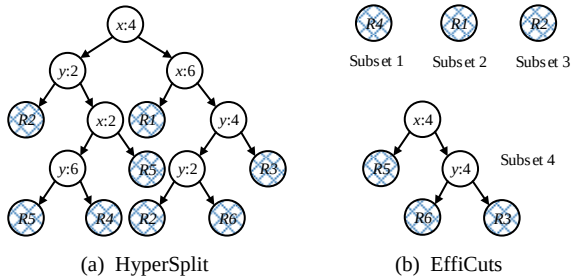
HiCuts[11], HyperCuts[12] and HyperSplit[4] are typical examples of such algorithms. HiCuts and HyperCuts apply equal-sized cutting to decompose the current space at each stage into equal-sized sub-spaces. In contrast, HyperSplit applies unequal-sized binary splitting at each stage, which avoids the inefficiency of equal-sized cuttings for non-uniformly distributed rules, and achieves superior performance compared to HiCuts and HyperCuts.

However, it is important to note that all these algorithms inevitably introduce rule replication in the decomposition steps, which may significantly increases the decision tree size. For example, in the HyperSplit decision tree illustrated in Figure 2(a), R2 and R5 are inevitably replicated once, causing memory overhead. As a matter of fact, the memory consumption of the decision tree grows *exponentially* as the size of ruleset or the number of dimensions increases.

C. Ruleset Partitioning Solutions

In recent years, several solutions based on *ruleset partitioning* [3][7][8][9] have been proposed to tackle the memory overhead issue.

In ruleset partitioning solutions, the original ruleset is divided into several subsets according to particular heuristic information; each of the sub-ruleset is then applied a traditional packet classification algorithm independently. In general, the sum of the sizes of the data structures built from each subset individually can be far smaller than the size of the data structure built from the full ruleset. This effective divide-and-conquer strategy makes it possible for the complex classification data structures to meet the overall memory consumption constrains again.



(a) HyperSplit (b) EffiCuts

Figure 2. Decision trees for the example ruleset

According to different ruleset partitioning methods, existing solutions can be categorized into structural characteristic based and stochastic search based.

Structural characteristic based partitioning: This kind of algorithms partitions the original ruleset according to the distribution or overlap characteristics of rules, in order for the rules in each sub-ruleset to have relatively nice "separability". As a representative algorithm of this category, EffiCuts[7] defines the rules in a subset to be separable if all the rules are either small or large in each dimension, and partitions the original ruleset by separating rules with different combinations of wildcard size (large or small) in all the dimensions. As shown in Figure 2(b), the original ruleset is partitioned into 4 groups according to this principle.

Stochastic search based partitioning: As one of the latest works, ParaSplit[9] views ruleset partitioning as a combinatorial optimization problem and applies simulated annealing to optimize the grouping of rules.

Both EffiCuts and ParaSplit achieve lower memory consumption than simply using a traditional packet classification algorithm (e.g., HyperSplit) alone. However, EffiCuts partitions the ruleset into $\Theta(2^D)$ groups, which is not scalable when the number of dimensions rises. And the selected tree merging step brings great uncertainty in performance and may introduce considerable rule replication in some cases. ParaSplit also suffers the downside of trapping into local optimum before the pre-defined limited number of iterations. It is because of the incapacity of simulated annealing in solving such combinatorial optimization problems with large and complex solution space. In general, the potential of ruleset partitioning, for minimizing the memory consumption of packet classification algorithms, has not been fully exploited.

III. PRELIMINARIES

A. Problem Modeling

In our approach, since the group number always follows the maximum number that a particular platform supports to execute in parallel (e.g., 8 cores or threads), it is given (determined) before the partitioning process.

Given a ruleset $RS = \{r_1, r_2, \cdots, r_N\}$ and the group number K, the aim is to find K disjoint subsets that minimize the overall memory consumption. Mathematically, it can be viewed as an integer programming problem:

$$\begin{aligned} &\min M(r_1, r_2, \cdots, r_N) \\ &s.t. \quad 1 \le r_i \le K \quad (i = 1, 2, \cdots, N) \\ &\qquad r_i \in Z \quad (i = 1, 2, \cdots, N) \\ &\qquad C\{r_i = k\} > 0 \quad (k = 1, 2, \cdots, K) \end{aligned}$$

where $r_i = k$ means that r_i is distributed in the k^{th} subset, and $M(r_1, r_2, \cdots, r_N) = \sum_{m=1}^{K} T(\bigcup_{r_i = m} r_i)$ represents the sum of the sizes of the data structures built from each subset, i.e., the overall memory consumption.

According to the principle of inclusion-exclusion, there are in total $\frac{1}{K!}\sum_{i=0}^{K}(-1)^i C_K^i (N-i)^N$ distinct feasible solutions to the partitioning problem. For example, there are about 10^{90} possible ways of grouping for the case N = 100, K = 8. Furthermore, the curve of $M(r_1, r_2, \cdots, r_N)$ is not smooth (e.g. moving only one rule from a group to another may be able to results in considerable memory increment/reduction in some cases). Therefore, it is difficult or even infeasible for brute-force computation or other straightforward methods to search for the optimal solution.

B. Grouping Penalty of Rule Pairs

To provide guidance to the rule grouping process of the proposed algorithm (described in the next section), we define the grouping penalty for two different rules. For a ruleset $RS = \{r_1, r_2, \cdots, r_N\}$, the grouping penalty of each rule pair is calculated as follows:

First, for each rule r, every dimension of r is labeled to be either *small* or *large* according to [7]. For the example of Figure 1, the six rules are labeled {*small*, *large*}, {*large*, *small*}, {*large*, *large*}, {*small*, *small*}, {*large*, *large*}, {*large*, *large*}.

Second, for each rule pair r_i and r_j, if in the d^{th} dimension the two rules are both *small* or both *large*, the size-conflict $C_{size}(d, r_i, r_j)$ is set to be 0; otherwise 1. If in the d^{th} dimension the ranges of the two rules are disjoint, the position-conflict $C_{pos}(d, r_i, r_j)$ is set to be 0.5; otherwise 1. The grouping penalty for r_i and r_j is calculated as follow:

$$P_m(r_i, r_j) = (\sum_{d=1}^{D} C_{size}(d, r_i, r_j) \cdot C_{pos}(d, r_i, r_j))^2 / D^2$$

where D is the number of dimensions.

IV. The SwinTop Algorithm

Swarm intelligence optimization algorithms, which simulate the collective behavior of natural systems (especially biological systems), have made great progresses on solving combinatorial optimization problems [13].

Since the ruleset partitioning problem is formulated into an integer programming problem with a huge and unsmooth solution space, it is very suitable to apply swarm intelligence algorithms with the characteristics of high searching capability, easy operation, and no special requirements for optimized function. Based on these principles, a revised hybrid swarm optimization algorithm *SwinTop*, which combines the best of Particle Swarm Optimization (PSO) and Genetic Algorithm (GA), is proposed to solve the ruleset partitioning problem.

In this section we present the details of SwinTop. Some optimizations to the implementation of SwinTop will be discussed in Section V.

A. Introduction to PSO and GA

1) Particle Swarm Optimization (PSO)

The basic concept of Particle Swarm Optimization (PSO) stems from the research on foraging behavior of bird blocks. PSO was first proposed by Eberhart and Kennedy in 1995 [14], since when PSO has been applied to numbers of fields including combinatorial optimization and data mining, etc.

Imagine a flock of birds seeking for one piece of food in a huge searching area. None of the birds knows the exact location of the food, but they do know the approximate distance between their positions and the food. As a matter of fact, the complex but intelligent global behaviors of birds are actually caused by the interactions of simple rules. By leveraging the equilibrium between the diversification and centralization, the bird flock can eventually find the food.

In a PSO algorithm, the location of the food represents the global optimum solution. The birds keep updating (optimizing) their speed and location through both competition and cooperation, until someone finding the food. As the crucial part of PSO, the three key elements that influence the update of a bird's speed include:

***Inertia*:** the bird keeps its previous speed in some degree.

***Self-cognition*:** the bird flies partly towards the best location that the bird itself has ever found.

***Social-cognition*:** the bird flies partly towards the best location that the entire flock has ever found.

2) Genetic Algorithm (GA)

Inspired by Darwin's biological theory of evolution, in 1975 J. Holland et al. proposed the first Genetic Algorithm (GA) [15], which simulates the mechanism of "the survival of the fittest" in biological evolution. The algorithm gained extensive attentions with the development of computer science, and was widely applied in fields like optimal control, pattern recognition and machine learning, etc.

GA imitates the co-evolutionary process of a *population* formed by multiple *individuals* (i.e., the locations of the birds in the context of PSO), and keep improving the fitness of the population until the strongest individual is found. A typical GA includes the following steps:

***Encoding*:** encode a feasible solution, i.e. an individual, into a chromosome according to the specific problem;

Initialization: initialize the individuals to form a population;

***Evaluation*:** evaluate all individuals at current iteration step and terminate the algorithm if the best individual is found;

***Selection*:** select some of the individuals according to certain probability model, promising that the excellent individuals are more likely to be selected;

Crossover and ***mutation:*** conduct chromosome crossover and mutation operation to the selected individuals and then return to the evaluation step.

B. Design Decisions of SwinTop

Inspired by PSO and GA, the proposed SwinTop algorithm simulates a bird flock seeking for food, while at the same time conducting gene (position) exchange and mutation selectively. The main design decisions include:

The combination of PSO and GA: Dealing with the multi-dimensional ruleset partitioning model, GA suffers from low convergence speed due to the randomness of crossover and mutation operations, but can guarantee global optimum when adopting the elitist strategy [16]. In contrast, PSO converges speedily but may trap into local optimum due to the lack of solution perturbation. In SwinTop, the best of PSO and GA are combined to keep the good convergence accuracy and speed simultaneously.

The introduction of individual pair compatibility: According to the grouping penalties of rule pairs, SwinTop can dynamically estimate the "compatibility" of two individuals (which will be explained later) to provide guidance to the iterations, and thus can further increase both the convergence accuracy and speed.

In the following part, the SwinTop algorithm will be presented in the order of *encoding*, *initialization*, and *iteration*.

C. Encoding

As described in Section III, when partitioning a ruleset with N rules into K subsets, the objective of SwinTop is to find the best distribution of rules $\{r_1, r_2, \ldots, r_N\}$ that minimizes the overall memory consumption. Table III shows the correspondence of some concepts and encodings.

TABLE III. CORRESPONDENCE OF CONCEPTS

Problem Concepts		**Biological Concepts**	
Integer programming	***Encoding***	***Classic PSO***	***Classic GA***
A variable in a feasible solution	r_i (the ID of the sub-ruleset that this rule is distributed into, ranging from 1~K)	Location of one dimension	Gene
A feasible solution	$\vec{r} = (r_1, r_2, \ldots, r_N)$	Location (Bird)	Chromosome (Individual)
Some feasible solutions	$\vec{r}_1 = (r_{11}, r_{12}, \ldots, r_{1N})$ ..... $\vec{r}_M = (r_{M1}, r_{M2}, \ldots, r_{MN})$	Bird flock	Population
The optimal solution	$\vec{r} = (r_1, r_2, \ldots, r_N)$ that minimize $M(r_1, r_2, \cdots, r_N)$	Location of the food	The strongest individual that maximize *fitness*

D. Initialization

In SwinTop, the population (i.e. bird flock) is initialized with 17 individuals (i.e. birds). Among them 8 individuals are called *pilots*, which will update themselves in each iteration step. Another 8 individuals are called *pbest_holders*, responsible for holding the historical best position of each pilot. The rest one individual is called *gbest_holder*, which holds the historical best position of all the pilots (i.e. *gbest* is the best of *pbests*).

To provide a nice initial solution, we propose a new direct ruleset partitioning algorithm (see Algorithm I) that approximates a relatively nice grouping of rules, which to some extent reduces the overall memory consumption and possesses the potential for further optimization by iteration.

ALGORITHM I. ALGORITHM FOR INITIAL PARTITIONING

```
function InitPartitioning (Ruleset)
1   Sub_Rulesets = {}
2   while Ruleset is not empty:
3     RS = {}
4     for each rule in Ruleset:
5       If rule is neither crossed nor partially-overlapped with any rule
      in RS:
6         RS.insert(rule)
7         Ruleset.delete(rule)
8     Sub_Rulesets.insert(RS)
9   return Sub_Rulesets
```

The basic idea of this direct partitioning algorithm is to eliminate all the orthogonal structures of rules. According to [17], orthogonal structure is the major (and also commonly encountered) pattern that causes a large amount of rule replication, leading to significant memory overhead.

For the initial population, one of the pilots is initialized with the result of the above algorithm, and the other pilots are initialized with random locations and random speeds. According to our observation, the special pilot is much likely to be the best solution among all the initial solutions.

E. Iteration

After initialization, the population begins to evolve. To evaluate each individual based on the objective function $M(r_1, r_2, \cdots, r_N)$, we define the *fitness* of individual as follow:

$$fitness(\vec{r}) = (m\_node \cdot 2N)^2 / (\sum_{m=1}^{K} T(\bigcup_{r_i = m} r_i))^2$$

where *m_node* is the memory size of each node in the decision tree built by a traditional packet classification algorithm, and T represents the actual size of a decision tree. In SwinTop, we choose HyperSplit to be the traditional algorithm for its superior memory efficiency compared to others, in which case *m_node* is 8 (Bytes).

The fitness ranges from 0 to 1. For an individual, the less memory overhead its grouping causes, the higher fitness it achieves. For the limiting case that no rule replication occurs (even for the default rule), the fitness reaches 1.

Next we define the compatibility of two individuals as follow:

$$compatibility(\vec{r}_1, \vec{r}_2) = K / \sum_{k=1}^{K} P_m(r_{1(k)}, r_{2(k)})$$

where P_m is the grouping penalty as defined in Section III, and $r_{1(k)}$ ($r_{2(k)}$) is a rule randomly picked from the k^{th} sub-rulesets of the feasible solution $\vec{r}_1$ ($\vec{r}_2$), i.e., $r_{1(k)} = r_{2(k)} = k$.

Compatibility is used to estimate the structural similarity between two individuals. The more compatible the two individuals are, the more likely that the distribution of their rules in each subset falls into similar patterns. By

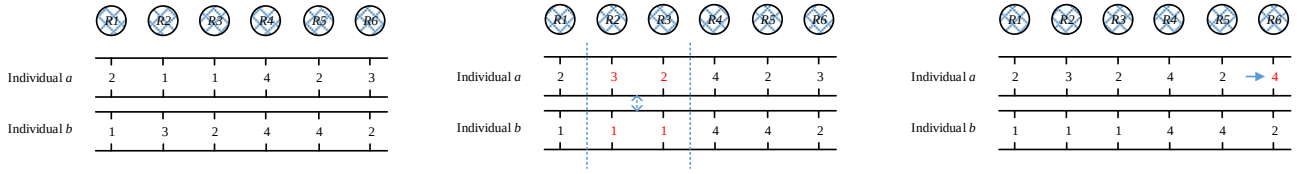
continuously checking the compatibility between an individual and current *gbest*, the algorithm can determine whether to conduct speed/position updates, or to conduct gene communications with other excellent individuals (including gbest), to evolve the individual itself. When compatibility is not enough, the former operations are chosen for encouraging the individual to bravely search in a broader area while to some extent drawing closer to the flock; otherwise, the latter is chosen, for making the individual carefully perturb current solution according to the effective information of others

mutation *p_mutation* = 0.2. In line 15, the Roulette Wheel Selection strategy is used to select one of the *pbests* based on their fitnesses (i.e., fitness-proportionate selection).

When any of these two criteria reached, the iteration terminates: (1) the fitness of *gbest* reaches 1, (2) the fitness of *gbest* has not increased for a certain number of iteration steps.

V. Optimization to SwinTop Implementation

During the iterations, the time-consuming packet classification algorithm (HyperSplit in our case) is frequently



(a) An example of two individuals. For instance, R1 is distributed in subset No.2 for the grouping represented by individual *a*.

(b) Crossover: randomly choose a part (e.g. (R2, R3)) and swap the genes of the two individuals in this part.

(c) Mutation: randomly choose a position (e.g. R6) and mutate the gene in this position.

Figure 3. Crossover and mutation

The pseudo-code of SwinTop algorithm is as below:

Algorithm II. The SwinTop Algorithm

```
function SwinTop (Ruleset)
1   Encode()
2   Initialize()
3   Calculate_fitness()
4   while termination condition not reached:
5     Update_pbests()
6     Update_gbest()
7     for each pilot in the population:
8       if pilot.fitness decreased and compatibility(pilot, gbest) not
      improved:
9         Update_speed(pilot)
10        Update_location(pilot)
11      else:
12        if rand(0,1) < p_crossover:
13          Cross(pilot, gbest)
14        if rand(0,1) < p_crossover:
15          Selected_pbest = Select(pbests)
16          Cross(pilot, selected_pbest)
17        if rand(0,1) < p_mutation:
18          Mutate(pilot)
19    Calculate_fitness()
20  return gbest
```

For line 9 and line 10, the methods of updating the speed and location is defined as follow:

$$v_i^{(n+1)} = c_0 \cdot v_i^{(n)} + c_1 \cdot rand(0,1) \cdot (pbest_i - v_i^{(n)}) + c_2 \cdot rand(0,1) \cdot (gbest_i - v_i^{(n)})$$
$$r_i^{(n+1)} = r_i^{(n)} + v_i^{(n+1)} \quad i = 1, 2, \cdots, N$$

where we set the inertia weight c_0 = 1, the self-cognition weight c_1 = 1.5, and the social-cognition weight c_2 = 0.9.

For line 12 to line 18, the crossover and mutation operations are illustrated in Figure 3. We set the possibility of crossover *p_crossover* = 0.5 and the possibility of

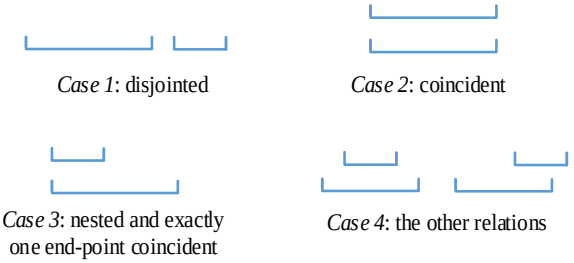


Figure 4. The four types of relations for two ranges

called to compute the accurate memory consumption for evaluating the fitnesses of individuals, resulting in significant amount of computation time before convergence. To tackle this issue, we propose a new concept, named *overlapping degree*, to estimate the memory consumption trends of rulesets much more efficiently.

The memory overhead of decision trees mainly comes from the overlap of rules [4]. We categorize the overlapping relations of two ranges into four types according to different possibilities of causing rule replication, as shown in Figure 4. We first define the overlapping degree of two rules:

$$ovlp\_r(r_i, r_j) = \sum_{d=1}^{D} seg(r_i, r_j, d) \cdot w(d)$$

where

$$seg(r_i, r_j, d) = \begin{cases} 0 & if\ case\ 1\ in\ the\ d^{th}\ dimension \\ 1 & if\ case\ 2\ in\ the\ d^{th}\ dimension \\ 2 & if\ case\ 3\ in\ the\ d^{th}\ dimension \\ 3 & if\ case\ 4\ in\ the\ d^{th}\ dimension \end{cases}.$$

For a ruleset, $w(d)$ is proportional to the number of unique end-points if all rules are projected on the d^{th}

dimension, and $\sum_{d=1}^{D} w(d) = 1$. The *ovlp_r* values are computed and stored in the initialization procedure, and will be directly read during iterations.

Next, the overlapping degree of a ruleset is defined as follow:

$$ovlp\_g(RS) = \frac{1}{C^2_{size(RS)}} \cdot \sum_{r_i, r_j \in RS,\ r_i \neq r_j} ovlp\_r(i, j)$$

which is the average of the *ovlp_r* of all the rule pairs. Thus in SwinTop, the objective function (i.e., the accurate overall memory consumption) can be replaced by:

$$\hat{M}(r_1, r_2, \cdots, r_N) = \sum_{m=1}^{K} overlap\_g(\bigcup_{r_i = m} r_i)$$

When implementing SwinTop, the estimation objective function $\hat{M}$ is first applied in the early stage of iterations, for efficiently finding a near-optimal grouping of rules. The accurate objective function M is then applied for acquiring the optimal grouping.

VI. EVALUATION

A. Data Set and Test-bed

The effectiveness and performance of the proposed SwinTop approach is evaluated using publicly available five-dimensional packet classification rulesets from ClassBench [5]. We use in total 18 rulesets, each of which is named according to its type (ACL: access control list, FW: firewall, IPC: IP chains) and size (100, 1K, 5K, 10K, 20K, 50K).

On an HP Z220 SFF workstation with 3.40GHz CPU, 16GB memory and 64bit Ubuntu 12.04, we implement SwinTop along with the two state-of-the-art partitioning algorithms, EffiCuts and ParaSplit, for comparison. The number of sub-rulesets is set to 8 for SwinTop and ParaSplit in all cases. For EffiCuts the number depends on the selective tree merging strategy and usually exceeds 8.

B. Effects of Combining PSO and GA

The iteration design of SwinTop is based on the revision and combination of classic PSO and classic GA. To verify the effects of this strategy, we implement the two classic optimization algorithms as well, using the same parameters as SwinTop (e.g., population size, crossover/mutation possibility, inertia/self-cognition/social-cognition weight, etc).

On each of the rulesets ranging from 100 to 5K, the three intelligent optimization algorithms (PSO, GA and SwinTop) are applied with the same initial population. We record the memory consumption of current best grouping when the iteration reaches 500 steps, 5000 steps, as well as when it meets the termination criteria.

In Table IV, the experimental results compared with classic PSO and GA manifest that the revised and hybrid swarm intelligent algorithm improves both the efficiency and accuracy of convergence. Specifically, in the early stage of iterations (see the memory consumptions of the 500th step), SwinTop searches for better solutions as fast as PSO does due to the target oriented strategies (i.e., location updates or crossover with excellent individuals); In the later stage of iterations (see the memory consumptions of the 5000th step and TERM.), SwinTop successfully avoid trapping into local optimum (which PSO is much likely to encounter due to the lack of solution perturbation). As the basis of the combination of PSO and GA, the introduction of "compatibility" enables the algorithm to adaptively switch between the two strategies during iterations.

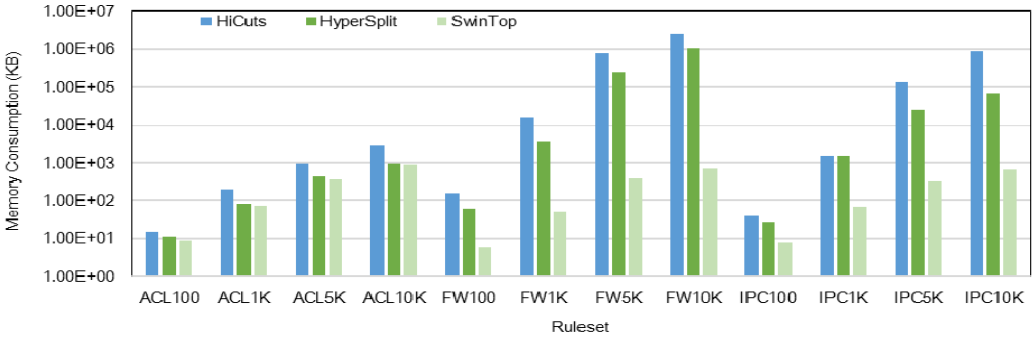


Figure 5. Memory consumption: SwinTop vs. traditional packet classification algorithms

TABLE IV. MEMORY CONSUMPTION IN DIFFERENT STATES OF ITERATIONS (KB)

Algorithm	*# of iterations*	*ACL*		*FW*		*IPC*	
		1K	*5K*	*1K*	*5K*	*1K*	*5K*
Classic PSO	500	139	1312	282	1720	146	1308
	5000	80	429	56	799	84	615
	TERM.	78	433	53	455	83	405
Classic GA	500	158	1380	319	1774	157	1341
	5000	79	483	55	1018	84	670
	TERM.	75	379	51	392	70	351
SwinTop	500	135	1320	243	1713	134	1312
	5000	81	428	54	682	83	586
	TERM.	75	368	51	375	70	324

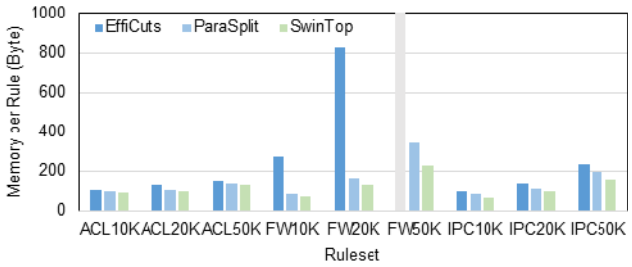


Figure 6. Memory per Rule: SwinTop vs. EffiCuts vs. ParaSplit

C. *Memory Efficiency*

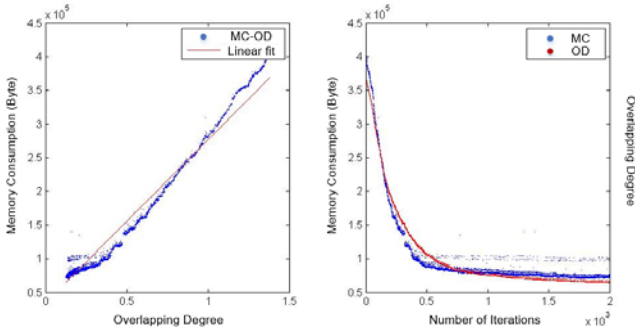
The memory performance of SwinTop is first compared against two representative packet classification algorithms, HiCuts and HyperSplit, in terms of memory consumption. In our workstation with only 16GB memory, it is always infeasible to build the decision trees for rulesets larger than 20K when simply using HiCuts or HyperSplit alone. Thus only the 100~10K rulesets are used for this experiment. The results are shown in Figure 5. For ACL rulesets, the memory reduction of ruleset partitioning is not significant (because the ACL rules are not intensively overlapped [4]). For FW and IPC rulesets, SwinTop achieves 1 to 4 orders of magnitude lower memory consumption compared with HiCuts and HyperSplit. For example, instead of building one huge decision tree for FW10K, by pre-partitioning the ruleset into 8 subsets and building 8 decision trees accordingly, the overall memory consumption is reduced from over 1GB to 719KB, which can fit in small-but-faster kinds of RAMs.

For comparing memory performance with EffiCuts and ParaSplit, the larger-sized rulesets (20K and 50K) are also tested. As shown in Figure 6, SwinTop outperforms the other two algorithms on all the rulesets. On average, SwinTop requires 50% less memory than EffiCuts and 25% less memory than ParaSplit. It benefits from SwinTop's capability of finding the global optimum grouping of rules.

D. *Effectiveness of Memory Comsumption Estimation*

In Section V, overlapping degree (OD, for short) is introduced to approximate the trends of memory consumption. On the representative ruleset FW_1K, we use OD-oriented SwinTop to optimize grouping, while at the same time calculating the accurate memory consumption (MC) in each iteration step. As shown in Figure 7(a) and 7(b), OD and MC share similar trends (i.e., as OD dives, so does MC), and the linear fitting effect is preferable, which meets the hypothesis of nice estimation.

Figure 7(b) also indicates that the best OD does not represents the best MC. However, the goal of OD in not to precisely replace MC, it is to provide a rough but much more efficient way to lower the objective function for the early stage of iterations. According to our test, compared with using MC all along, the introduction of OD achieves on average 20-fold reduction in convergence time.



(a) Linear fitting of MC and OD (b) OD-oriented iteration

Figure 7. Effectiveness of memory consumption estimation

VII. CONCLUSION

To tackle the urgent memory utilization issue of packet classification function in network devices, this paper proposes SwinTop, a new ruleset partitioning approach for significantly reducing the size of memory consumed by the traditional packet classification algorithms. Based on the study of ruleset characteristics and the modeling of the ruleset partitioning problem, SwinTop develops a novel swarm intelligence based algorithm to seek for the global optimum grouping of rules that minimizes the memory consumption of packet classification decision trees. The design of SwinTop is extensible to future network functionalities with larger, more complex, and higher-dimensional rulesets used.

REFERENCES

[1] G. Pankaj, and N. McKeown, "Algorithms for packet classification," IEEE Transactions on Network, 2011.

[2] S. Haoyu, "Design and evaluation of packet classification systems," Doctoral dissertation, Washington University, Department of Computer Science and Engineering, 2006.

[3] Z. Kai, L. Zhiyong, and G. Yi, "Gear up the classifier: scalable packet classification optimization framework via rule set pre-processing," Proc. of the ISCC, 2006.

[4] Q. Yaxuan, X. Lianghong, Y. Baohua, X. Yibo, and L. Jun, "Packet classification algorithms: from theory to practice," Proc. of IEEE INFOCOM, 2009.

[5] http://www.arl.wustl.edu/~hs1/PClassEval.html

[6] Masoud Moshref, Minlan Yu, Abhishek Sharma, and Ramesh Govindan, "Scalable rule management for data centers," Proc. of NSDI, 2013.

[7] Balajee Vamanan, Gwendolyn Voskuilen, and T. N. Vijaykumar, "Efficuts: optimizing packet classification for memory and throughput," Proc. of ACM SIGCOMM, 2011.

[8] J. Weirong, V. K. Prasanna, and N. Yamagaki, "Decision forest: a scalable architecture for flexible flow matching of fpga," Proc. of IEEE Field Programmable Logic And Applications, 2010.

[9] F. Jeffrey, W. Xiang, Q. Yaxuan, and L. Jun, "ParaSplit: a scalable architecture on fpga for terabit packet classification," Proc. of IEEE High-performance Interconnects, 2012.

[10] M. H. Overmars, and A. F. van der Stappen, "Range searching and point location among fat objects," Journal of Algorithms, vol. 21(3), 1996.

[11] G. Pankaj, and N. McKeown, “Packet classification using hierarchical intelligent cuttings,” Proc. of Hot Interconnects, 1999.

[12] S. Singh, F. Baboescu, G. Varghese, and W. Jia, “Packet classification using multidimensional cutting,” Proc. of ACM SIGCOMM, 2003.

[13] http://en.wikipedia.org/wiki/Swarm_intelligence

[14] J. Kennedy, and R. Eberhart, “Particle Swarm Optimization,” Proc. of IEEE International Conference on Neural Networks, 1995.

[15] J. H. Holland, “Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence,” U Michigan Press, 1975.

[16] G. Rudolph. “Convergence analysis of canonoical genetic algorithms” IEEE Transaction on Neural Networks, 1994.

[17] H. Peng, X. Gaogang, K. Salamatian, and L. Mathy, “Meta-algorithms for software-based packet classification,” Proc. of IEEE International Conference on Network Protocols, 2014.