IET Communications

Research Article

IET Journals
The Institution of Engineering and Technology

Memory-efficient recursive scheme for multi-field packet classification

ISSN 1751-8628
Received on 20th October 2018
Revised 28th January 2019
Accepted on 27th February 2019
E-First on 29th April 2019
doi: 10.1049/iet-com.2018.6038
www.ietdl.org

Wenjun Li[1], Dagang Li[1,2], Yongjie Bai[1], Wenxia Le[3], Hui Li[1] ✉

[1]*School of Electronic and Computer Engineering, Peking University, University Town, Nanshan District, Shenzhen, People's Republic of China*
[2]*PKU-HKUST ShenZhen-HongKong Institution, Shenzhen High-Tech Industrial Park, Nanshan District, Shenzhen, People's Republic of China*
[3]*Network Energy Department, Huawei Technologies Co., Ltd, Huawei Base, Bantian, Longgang District, Shenzhen, People's Republic of China*
✉ *E-mail: lih64@pkusz.edu.cn*

Abstract: Multi-field packet classification is not only an indispensable and challenging functionality of existing network devices, but it also appears as flow tables lying at the heart of the forwarding plane of software defined networking age. Despite almost two decades of research, algorithmic solutions still fall short of meeting the line-speed of high-performance network devices. Although decomposition-based approaches, such as cross-producting and recursive flow classification (RFC), can achieve high lookup rate by performing a parallel search on chunks of the packet header, both of them suffer from memory explosion problem during aggregation. In this study, the authors propose an HybridRFC, a memory-efficient recursive scheme for multi-field packet classification. By addressing the embedded problem of the RFC caused by uncontrollably expanded cross-product tables, HybridRFC can not only reduce the memory consumption to a practical level but also improve pre-processing performance significantly. Experimental results show that the memory requirement of HybridRFC is two orders of magnitude less than RFC, as well as three orders of speed-up on the performance of table building on average.

1 Introduction

Modern network devices provide services beyond basic packet forwarding, such as access control, security, policy routing and quality of service (QoS). Multi-field packet classification is the core functionality for supporting these services, which decides the action to be taken on a packet based on multiple fields in the packet header. A predefined classifier consisting of a set of rules is looked up for a match for these purposes. To match a rule, packet classification needs to compare multiple header values of the incoming packet against the field values of all the rules in the classifier to determine the type of actions (e.g. drop or permit) to be taken on the packet. With the adoption of software-defined networking (SDN), packet classification has become even more prominent than ever, where OpenFlow plays a central role in the forwarding plane of SDN [1]. An example OpenFlow flow table is shown in Table 1. Although the recent OpenFlow specification puts forward the multiple match tables (MMT) model that allows multiple smaller flow tables to be matched in a pipeline of stages, MMT still contains some complex flow tables and needs to conduct fast packet classification. Recently, P4 has been introduced to support programming protocol independent packet processing, where a compiler transforms an imperative program into a table dependency graph that can be mapped to a pipeline model with multiple ordered tables with an arbitrary number of fields. Thus, multi-field packet classification is still a key functionality in these new configurable switch architectures. Due to its importance and challenge, packet classification has attracted research attention for almost two decades.

Packet classification is a challenging problem due to the line-speed requirement of network devices, where a packet has to be processed within a very short time. The common practice for line-speed packet classification in the industry is to use ternary content addressable memory (TCAM), which is a fully associative memory that allows a 'don't care' state to be stored in each memory cell in addition to 0 s and 1 s. When a packet is presented to the TCAM, each TCAM entry is looked up in parallel and the best matching rule is returned. Thus, a single TCAM access is sufficient to perform a packet classification. Although TCAM enables parallel lookups on rules for line-speed classification, this brutal force hardware solution is not only expensive, but also very power-hungry, which seriously limits its scalability. During the past decade, a lot of methods and algorithms had been proposed to alleviate these problems, such as classifier minimisation [2–6], range encoding [7–11] and pre-classifier [12–15]. However, due to inherent limitations of TCAM, the TCAM capacity is not expected to increase significantly in the near future [4].

Recently, researchers have been actively investigating algorithmic alternatives for TCAM, such as hash-based algorithms [16, 17], decision-tree techniques [18–26] and decomposition-based schemes [22, 27–29]. Among them, techniques based on decomposition have been recognised as some of the most promising approaches for designing high-throughput network devices, since they can leverage the parallelism offered by modern hardware to accelerate lookup performance. Cross-producting [30], a classic decomposition algorithm, can scale well with regard to the number of fields as well as the type of field specification. However, cross-producting will build up a big cross-product table

Table 1 Example OpenFlow 1.0 classifier

Rule	Ingress port	Ether src	Ether dst	Ether type	VLAN id	VLAN priority	IP src	IP dst	IP proto	IP ToS bits	TCP/UDP src port	TCP/UDP dst port	Action
R_1	3	*	*	2048	*	*	206.159.213.125/32	101.152.182.8/30	0x06f/0xff	0	1024:65535	*	forward
R_2	5	*	*	2048	*	*	15.25.70.8/30	*	0x11f/0xff	0	*	0:1599	enqueue
...	...	...	...	...	...	...	...	...	...	...	...	...	...
R_n	*	*	*	*	*	*	*	*	*	*	*	*	drop

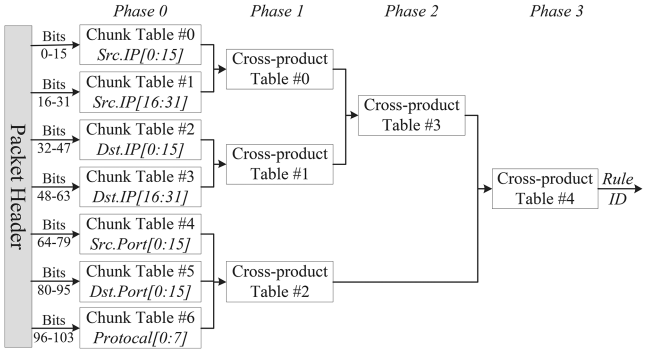

Fig. 1 *Example of classic 5-tuple packet flow in RFC (#phase=4)*

that requires excessive memory capacity. In order to alleviate the problem, recursive flow classification (RFC) [31], performs a search in multiple phases instead of a single phase, so that much smaller cross-product tables will be needed in each aggregation phase. Nevertheless, with the expansion of the rule set, the memory consumed in an RFC will still increase drastically and cause memory explosion. Instead of splitting all fields into chunks for aggregation, hierarchical space mapping (HSM) [29] reduces the searching fields by mapping the lookup domains two-to-one, step by step and hierarchically. To solve the memory explosion problem, distributed crossproducting of field labels (DCFL) [28] proposes a hardware-based distributed cross-producting scheme working with field labels. However, it is difficult to be widely used in industry because the assumptions in integrated circuit technology advancement of DCFL are unrealistic. To speed up the table building, fast table building for recursive flow classication (FRFC) [27] divides the total rule set into multiple sub-sets and creates tables for each sub-set with RFC individually. HybridCuts [22], a recent proposed decomposition-based cutting scheme, can improve storage and performance simultaneously by partitioning rules into subsets. However, its partition scheme is based on the observations about typical 5-tuple classifiers, which seriously limit its adaptability for general flow tables. As an improvement, CutSplit [21] partitions rules based on general small field rather than address fields, but its performance is undetermined and depends upon the specific rule bases. As far as we know, most of the existing decomposition-based approaches can be well applied for some modest classifiers, but still suffer from memory explosion for large classifiers. All in all, memory is still a precious resource in current network packet processing devices. Reducing memory consumption can not only reduce the cost but also accelerate classification by putting rules into static random-access memory (SRAM).

In this paper, we propose HybridRFC, a memory-efficient decomposition-based scheme for multi-field packet classification. We first seek to understand the reasons behind the memory explosion problem suffered by RFC algorithm. Then we make some important observations which provide a good substrate for our proposed algorithm. Finally, on the basis of the idea of saving sparse CBM (class bitmap) entries from further accessing to the subsequent aggregations, we present our proposed HybridRFC, which can reduce the memory consumption to a practical level even for large classifiers. In addition, by separating rules into subsets and shuffling rule bits individually, HybridRFC can also improve pre-processing performance significantly. Experimental results show that using ClassBench [32], HybridRFC achieves a memory reduction of 268.8 times compared to RFC, as well as 965.2× speed-up on the performance of table building on average. Compared to CutSplit, the state-of-the-art algorithmic approach, HybridRFC also achieves an average of 2.6× improvement on performance in terms of the number of memory access. Due to its high performance and good scalability in arbitrary number of fields, HybridRFC can not only support fast packet classification in traditional network devices, but also be well applied to the new configurable switch architectures such as MMT and P4.

The rest of this paper is organised as follows. In Section 2, we first give some background and briefly summarise our goals. After that, we make a set of key supporting observations and present the technical details of HybridRFC in Section 3. Section 4 verifies the effectiveness of HybridRFC with experimental results. Finally, Section 5 draws conclusions on this work.

2 Background

2.1 Multi-field packet classification problem

The purpose of multi-field packet classification is to find a matching rule from a predefined packet classifier for a packet. A packet classifier is a set of rules, with each rule R consisting of a tuple of F field values (exact value, prefix or range) and an action to be taken in case of a match. For example, a typical 5-tuple rule consists: the source and destination IP addresses (i.e. SA, DA), the source and destination ports (i.e. SP, DP), and the protocol number (i.e. *Prot*). The rules are often prioritised to resolve potential multiple match scenarios.

Packet classification can be treated as a point location problem in computational geometry and it has been proved that the best bounds for locating a point are either $\Theta(\log N)$ time with Θ (N ^{F}) space, or $\Theta(\log^{N} F-1)$ time with Θ(N) space for N non-overlapping hyper-rectangles in F-dimensional space [24]. Clearly, this is impractical even for small classifiers. But fortunately, packet classification rules in real-life applications have some inherent characteristics that can be exploited to reduce the complexity. These inherent characteristics provide a good substrate for the exploration of practical algorithmic solutions. The following is a distillation of previous observations relevant to our work [8, 14, 21, 22, 24, 30, 31]: (i) the protocol field is restricted to a small set of values; (ii) rules specify a limited number of distinct transport port ranges; (iii) the number of address prefixes matching a given address is typically five or less; (iv) the number of rules matching a given packet is typically five or less; (v) many rules share the same field values; (vi) most of the rules have at least one small field; (vii) most of the rules have at least one small address field.

2.2 Related work and our goals

Since our proposed HybridRFC is designed mainly based on the well-known RFC algorithm, we next give a more detailed review on RFC in this subsection.

RFC algorithm [31], a generalisation of cross-producting [30], builds multiple lookup tables through multiple phases as illustrated in Fig. 1. In phase 0, all fields of the packet header are split into multiple chunks that are used for indexing memories. In subsequent phases, the cross-product tables are formed by a combination of several temporary CBM tables that are results of the lookups from previous phases. Each lookup will map the chunk to an 'chunk equivalence set' (CES) according to the rules. The term CES is used to denote a set of chunk values that have the same *eqID*, which is determined by the rules corresponding to this chunk. To facilitate the calculation of *eqIDs* for subsequent phases, the 'class Bitmap' (CBM) is used for each CES to indicate the rules that are contained in this CES for the corresponding chunk. Thus, there are two data in CBM tables: an identification number (i.e. *eqID*) of the CBM table and a bitmap for the matched rules (i.e. CBM). For each bitmap in CBM tables, the ith bit is set to bit-1 if the given key value is matched with the corresponding fields of the ith rule. In the final phase, the matched rule ID can be obtained from the lookup. For each incoming packet, it performs independent, parallel searches on chunks of the packet header recursively. Fig. 2 shows a sample construction of cross-product table for a classifier with eight rules (i.e. eight bits for each CBM).

From the above review, we can see that there are two main sources of memory consumption in RFC: chunk table and cross-product table. As we know, when the size of the rule set grows, the cross-product tables become the dominant cause for the memory explosion problem in RFC. Thus, the goal of our work is to solve the memory explosion problem in RFC by carefully controlling the rapid growth on the cross-product tables. Next, we give some more technical details of our proposed HybridRFC.

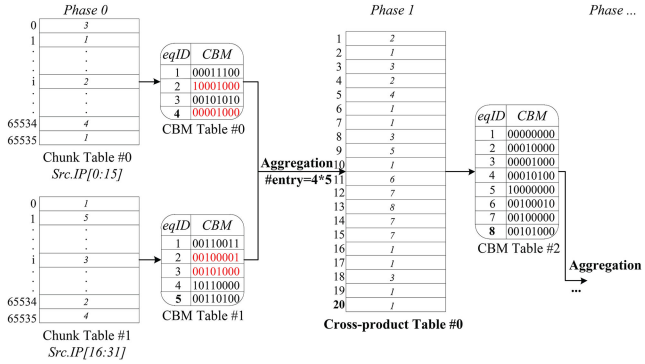

Fig. 2 *Sample construction of the cross-product table in RFC. Assuming the threshold as 2, four bitmaps marked in red are indicated as sparse CBM entries, which will be removed out for linear search in HybridRFC*

Table 2 Average ratio of bit-1 in CBM tables in different phases

rule set (#rule)	Phase 0	Phase 1	Phase 2	Phase 3	Mean
seed-ACL (752)	0.136	0.120	0.005	0.005	0.066
seed-FW (269)	0.459	0.241	0.080	0.010	0.197
seed-IPC (1550)	0.257	0.243	0.009	0.003	0.128
Mean	0.284	0.201	0.031	0.006	0.131

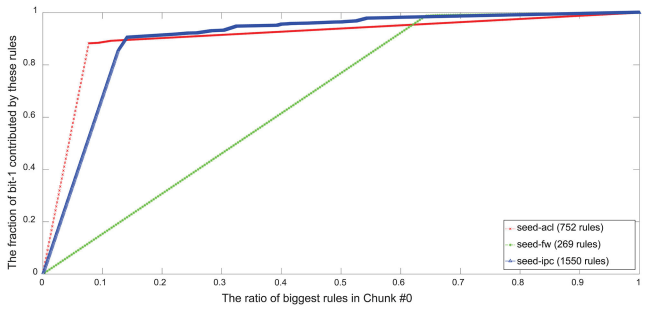

Fig. 3 *Influence of big rules for CBM table #0 in Fig. 1 for different rule sets*

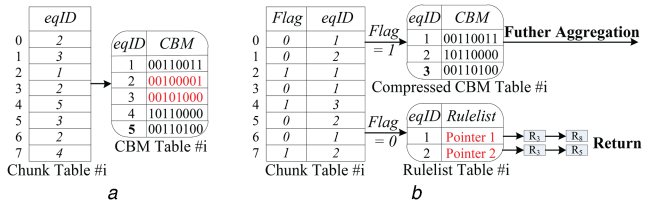

Fig. 4 *Example of data structure for tables in HybridRFC (threshold = 2)*
(a) Data structure in RFC, ***(b)*** Data structure in HybridRFC

3 HybridRFC

In this section, we first make some observations on the inherent characteristics of CBM tables in RFC. After this, based on these key supporting observations, we detail three key ideas which can help to reduce the memory consumption of the cross-product tables. Finally, we give the refined framework of HybridRFC.

3.1 Key observations

By examining the cross-product table construction for three different types of rule sets from ClassBench, we identify three following important and helpful observations on CBM tables in RFC.

Observation 1: Most of the class bitmaps in CBM table are relatively sparse. In other words, the number of rules matched by each CES is much smaller than the rule set size, where each CES is denoted by a class bitmap in CBM table. For example, there are (4+5+8)*8=136 bitmap bits in three CBM tables in Fig. 2, but only 33 bits are marked as 1. Thus, the average ratio of bit-1 in Fig. 2 is 0.24. More experimental results using ClassBench as shown in Table 2 possess similar sparseness, where cross-product tables are constructed as shown in Fig. 1. We can see that the mean ratio of bit-1 for three different rule sets is only 0.131.

Observation 2: At later aggregation phases, the above sparse feature of CBM will become more apparent. Take Fig. 2 as an example. The average ratio of bit-1 in phase 0 is 23/72=0.32 while it drops to 10/64=0.16 in phase 1. Table 2 shows more details in our experiments.

Observation 3: Big rule is the main contributor to bit-1 s in CBM tables. In other words, rules that contain big field range may be present in a large amount of CBM entries. Note that unlike previous definitions about big/small given in [21, 22], the notation of big described here is a more relative concept as the width of chunk is variable. For example, 192.168.*.* may be considered as a small field in [21, 22], but it is the biggest field (i.e. wildcard in all lower 16 bits) for chunk table #0 in Fig. 1. Based on these definitions and notations, Fig. 3 shows the influence of big rules for CBM table #0 from Fig. 1. We can see that more than 90% bit-1 s are marked by the less than 10% biggest rules for most rule sets. Considering that a significant fraction of FW rules have wildcard address fields (about 65% for the lower 16 bits), the results for seed-fw in Fig. 3 is still consistent with our observation.

3.2 Key ideas

Based on the above observations, we introduce three key ideas of HybridRFC, where the first one is the basis and the other two are optimisations. The rationale behind these ideas is simple: since each cross-product table in RFC is formed by a combination of two or more CBM tables from its previous phase, the idea directly perceived is to reduce the number of valid entries in the CBM tables, so that the total number of the following cross-product entries as well as the memory consumption will be reduced.

(1) *Remove sparse bitmap entries from the CBM table so as to reduce the number of the following cross-product entries.* As each cross-product table is formed by a combination of several CBM tables from its previous phase, and the final matching rule must be indicated by bit-1 in all matched bitmaps from different phases, the rules indicated by bit-0 in any intermediate CBMs cannot be the final matching rule. For example, if the key value in the packet matches the second bitmap entry in CBM table #1 in Fig. 2 (i.e. class bitmap = 00100001), we can be sure that only the third and eighth rule (i.e. R_3 and R_8 in Fig. 4) can be the final matching rule. Similarly, if the key value in the packet matches the third bitmap entry in CBM table #1 in Fig. 2 (i.e. class bitmap = 00101000), we can be sure that only the third and fifth rule (i.e. R_3 and R_5 in Fig. 4) can be the final matching rule. Since most of the class bitmaps in CBM tables are relatively sparse, we can simply conduct linear search on the rule-list which contains these rules indicated by bit-1 to find the best matching, rather than further aggregation with other CBM tables. To support this new search strategy, we can simply modify the data structure of chunk and cross-product tables by adding one flag bit. Fig. 4 shows more details of modified data structure of HybridRFC. Based on this idea, if we set the threshold at two such that at most two rules are allowed for linear search, we can remove four bitmap entries in phase 0 that marked in red from CBM table #0 and CBM table #1 in Fig. 2. Thus, the size of the following cross-product table #0 can drop down from 4*5 = 20 entries to 2*3 = 6 entries.

(2) *Shuffle rule bits to enhance the sparseness of CBM table entries in early phases.* Based on the second observation that the sparse feature of CBM will become more apparent at later aggregation phases, we can shuffle rule bits from discriminative fields to reduce the total number of cross-product entries in RFC. The rational behind this strategy is simple: by shuffling rule bits from discriminative fields, we can get more sparse bitmap entries that can be removed from CBM tables in early phases, so that the total number of the following cross-product entries as well as the memory consumption will be reduced. In essence, the second observation above is consistent with many previously discoveries

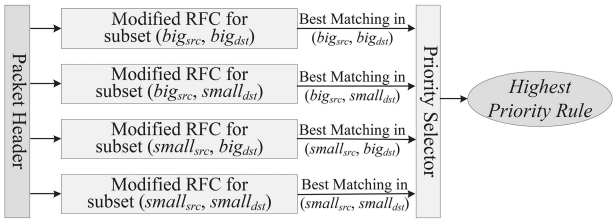

Fig. 5 *Framework of HybridRFC. Note that the middle four modified RFCs can be searched in parallel*

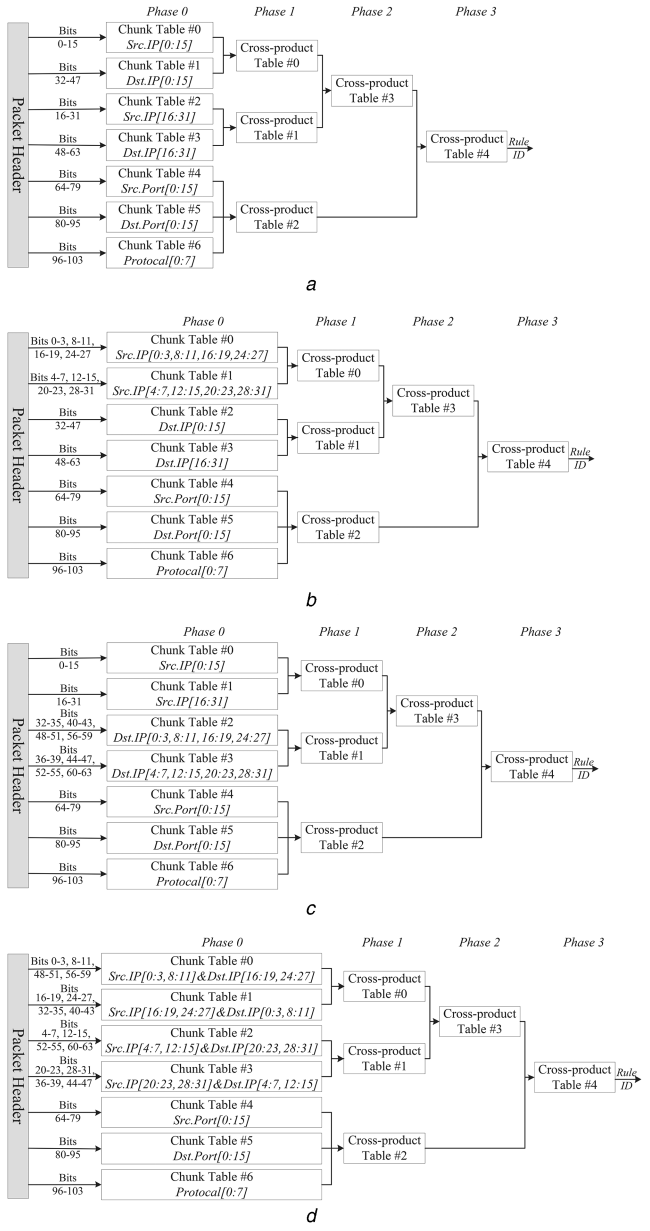

Fig. 6 *Modified RFCs for partitioned rule subsets in HybridRFC*
(a) Modified RFC for (big_{src}, big_{dst}), ***(b)*** Modified RFC for ($small_{src}$, big_{dst}), ***(c)*** Modified RFC for (big_{src}, $small_{dst}$), ***(d)*** Modified RFC for ($small_{src}$, $small_{dst}$)

about real-life rule sets, which has been well summarised in the above section. Three of those inherent characteristics are critical for our second idea: (i) The number of rules matching a given packet is not very large - typically five or less; (ii) The number of IP address prefixes matching a given packet is also typically five or less; (iii) Many different rules share the same field. Thus, we can conclude that, with more discriminative field combined, the sparseness of the corresponding CBM table will be more obvious. Based on this conclusion, we propose a modified RFC algorithm for each following partitioned rule subset, which shuffles the rule bits for each chunk, rather than mapping continuous bit ranges for each chunk as in RFC. For example, instead of splitting the lower 16 bits in the source address (i.e. *Src.IP[0:15]*) for chunk #0 in Fig. 1, we can instead choose 8 bits from the source address and 8 bits from the destination address (e.g. *Src.IP[0:3]*, *Src.IP [8:11]* and *Dst.IP[16:19]*, *Dst.IP[24:27]*). By shuffling address bits, we can obtain more sparse CBM tables in early phases, so that more CBM entries become sparse and get removed early in the classifier for linear search.

(3) *Partition rules into subsets to relieve influence of big rules, which will further enhance sparseness of CBM tables.* According to the third observation, since big rule/field is the main contributor to bit-1 s in CBM tables, separating big rules from small rules can enhance sparseness of CBM entries significantly. As we all know, rule set partitioning is an effective method that has been wildly used in many recent algorithmic approaches. Based on previous observation that IP address is the most significant field for packet classification [13, 14, 20, 22, 31], we can simply separate rules into the following four subsets based on address field length: (big_{src}, big_{dst}), (big_{src}, $small_{dst}$), ($small_{src}$, big_{dst}) and ($small_{src}$, $small_{dst}$). Obviously, partitioning may degrade lookup performance for most algorithmic approaches such as decision-trees, but it will not be a problem for decomposition based algorithms, since they can leverage the parallelism offered by modern hardware.

3.3 Framework

Based on above observations and key ideas, we now present the framework of HybridRFC as shown in Fig. 5. The modified RFC is designed based on above first two key ideas, which is different from RFC in two perspectives: remove sparse CBM entries and shuffle rule bits.

In order to reduce influence of big rules for CBM tables, we first separate the rule set into four subsets based on two IP address length. More details about rule set partition algorithm can refer to [21, 22]. For each subset, we then construct a modified RFC individually as shown in Fig. 6, where bits in small field are recombined with 4-bit interval in our experiments. The reason for this shuffle algorithm is based on the observation that small field is more discriminative than large field. Thus, with partial bits from small fields combined, the sparseness of the corresponding CBM table in early phases will be more obvious. During classification, each incoming packet will first search in 4 subsets in parallel and the matching rule with the highest priority will then be chosen as a result.

Of course, rule bit shuffle algorithm is an open problem. The specific algorithm described in Fig. 6 is a practical implement in our experiments, but not the optimal solution. Due to the consideration on its relevance, more optimisations on shuffle algorithm will be further studied in our future work.

3.4 Working example

To illustrate how HybridRFC works, we consider a simple working example based on the classifier in Table 3 which contains eight 2-tuple rules. For the simplicity of description, we assume there are only two types of values: wildcard (i.e. *) as big field and specific value as small field.

We first split the 16-bits into two 8-bits chunks to express how RFC works in two recursive phases. The field X column in Table 3 partitions the possible values into five sets (i.e. CES): (a) {*} (b) {00001111, *} (c) {01010101, *} (d) {10101010, *} (e) {11110000, *}; which can be indexed by *eqID* from *eqID* = 1 to *eqID* = 5. The field Y column in Table 3 also partitions the possible values into five sets: (a) {*} (b) {00001111, *} (c) {01010101, *} (d) {10101010, *} (e) {11110000, *}; which can be indexed by *eqID* from *eqID* = 1 to *eqID* = 5. For each CES, a CBM is assigned to indicate which rules contain this CES for the corresponding chunk. Fig. 7 shows the contents of RFC tables for the example classifier of Table 3. The sequence of accesses made by the example packet have also been shown using big gray arrows and the memory locations accessed in this sequence have been marked in bold. By separating rules into subsets based on small and big field, HybridRFC can partition the rule set into four subsets:(a)

Table 3 Example 2-tuple classifier (field width = 8)

Rule	Priority	Field X	Field Y	Action
R_1	8	10101010	10101010	$action_1$
R_2	7	10101010	01010101	$action_2$
R_3	6	01010101	10101010	$action_3$
R_4	5	01010101	01010101	$action_4$
R_5	4	11110000	00001111	$action_5$
R_6	3	00001111	********	$action_6$
R_7	2	********	11110000	$action_7$
R_8	1	********	********	$action_8$

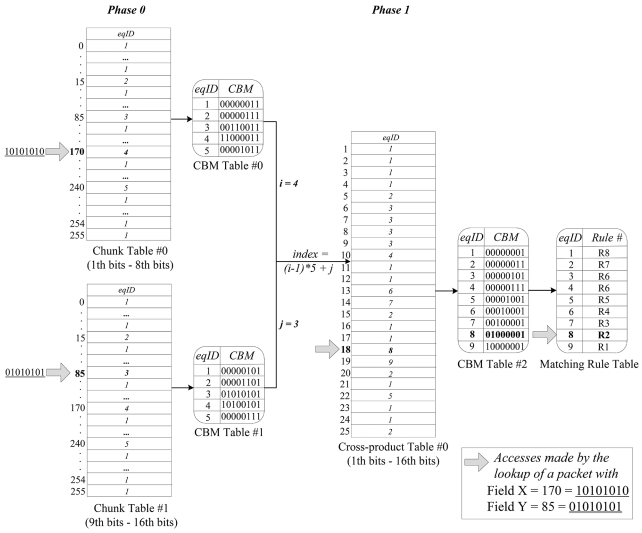

Fig. 7 *Contents of RFC tables for the example classifier*

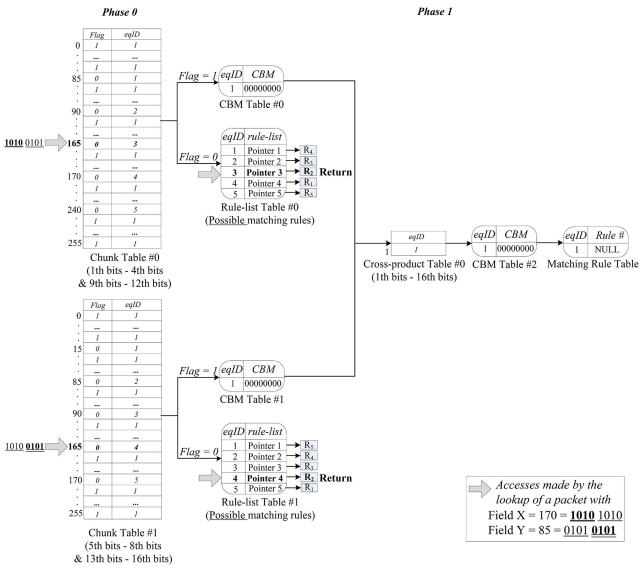

Fig. 8 *Contents of HybridRFC (threshold = 1) tables for the classifier* ($small_X$, $small_Y$)

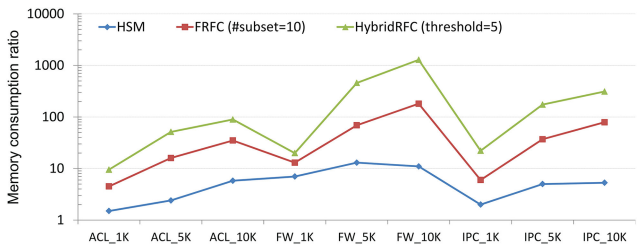

Fig. 9 *Reduction of memory consumption in comparison with that of RFC*

(big_X, big_Y) = {R_8} (b) (big_X, $small_Y$) = {R_7} (c) ($small_X$, big_Y) = {R_6} (d) ($small_X$, $small_Y$) = {R_1, R_2, R_3, R_4, R_5}. Fig. 8 shows the contents of HybridRFC tables for the partitioned rule subset: {R_1, R_2, R_3, R_4, R_5}. The sequence of accesses made by the example packet have also been shown using big gray arrows and the memory locations accessed in this sequence have been marked in bold. From these two figures, we can see that HybridRFC can significantly reduce the size of cross-product tables by shuffling rule bits and removing rules in sparse bitmaps for linear search.

4 Experiment results

In this section, we present the performance of HybridRFC with two other representative RFC based schemes: HSM and FRFC, as well as two latest algorithmic approaches: HybridCuts and CutSplit. We evaluate the performance with rule sets generated using ClassBench [32], whose size varies from 1 to 10 K. There are three types of rule sets: access control list (ACL), Firewall (FW) and IP chain (IPC). Each rule set is named by its type and size, e.g. FW_1K refer to the Firewall rule set with about 1000 rules. We evaluate our algorithm from memory consumption, pre-processing time and memory access respectively. Besides, we also evaluate the effectiveness of the key ideas in HybirdRFC. All experiments are run on a machine with AMD A8-5600 K CPU@3.6 GHz and 8G DRAM. The operation system is Ubuntu 14.04.

4.1 Memory consumption

Table 3 shows the memory consumption as well as pre-processing time of HybridRFC in comparison with that of the original RFC algorithm. Since the number of rules matching a given packet is typically five or less, we set the threshold at five such that at most five rules are allowed for linear search in HybridRFC. Experimental results show that HybridRFC achieves a significant memory reduction compared with the original RFC algorithm, ranging from 9.5 times to 1284.1 times, with an average reduction of 268.8 times. Even for large rule sets up to 10 K, the memory requirement of HybridRFC can still be limited to a few megabytes in our evaluations.

Fig. 9 shows the reduction ratio of memory consumption in comparison with that of RFC algorithm for HSM, FRFC and HybridRFC. Compared to HSM and FRFC, HybridRFC also achieves 45 × and 5 × improvement respectively.

4.2 Pre-processing time

Compared with the original RFC algorithm, HybridRFC achieves a significant speed-up on pre-processing performance as illustrated in Table 4, ranging from 2.1 times to 3991 times, with an average speed-up of 965.2 times. Even for large rule sets up to 10 K, HybridRFC can still finish table building in a few seconds, while it may cost a few hours in RFC.

Fig. 10 shows the improvement of pre-processing time in comparison with that of RFC algorithm for HSM, FRFC and HybridRFC. Compared to HSM and FRFC, HybridRFC also achieves 51 × and 4 × improvement respectively. Thus, by separating rules into subsets and removing sparse entries for individual processing, HybridRFC can not only reduce memory consumption to a practical level, but also improve pro-processing performance significantly.

4.3 Effectiveness of key ideas

To gain more insights, we make several evaluations on the effectiveness of the three key ideas in HybridRFC, which can be seen from Figs. 11 and 12. Each evaluated scheme is named by the key ideas it adopted, e.g. HybridRFC_idea1 refer to the improved RFC by only removing sparse bitmap entries from its CBM tables. Obviously, these key ideas really enhance the sparseness of CBM table entries, which in turn reduce the total number of the following cross-product entries.

Table 4 Effectiveness of HybridRFC compared to RFC

Rule set	Memory consumption		Pre-processing time	
	RFC, MB	HybridRFC, MB	RFC, s	HybridRFC, s
ACL_1K	12.4	1.3	0.63	0.3
ACL_5K	77.1	1.5	28	0.8
ACL_10K	187.5	2.1	187.6	2.1
FW_1K	33.8	1.7	9.2	0.4
FW_5K	1099.2	2.4	725.4	1.3
FW_10K	3980.7	3.1	37914.5	9.5
IPC_1K	37.6	1.7	37.8	0.7
IPC_5K	504.6	2.9	1168.2	3.3
IPC_10K	1803.8	5.8	31512.8	8.8
average reduction	268.8		965.2	

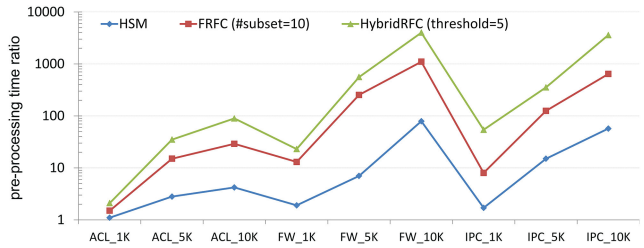

Fig. 10 *Reduction of pre-processing time in comparison with that of RFC*

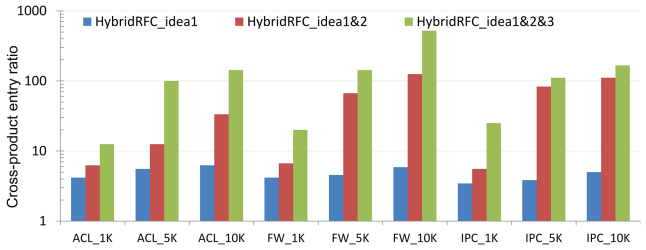

Fig. 11 *Reduction of cross-product table size in comparison with that of RFC*

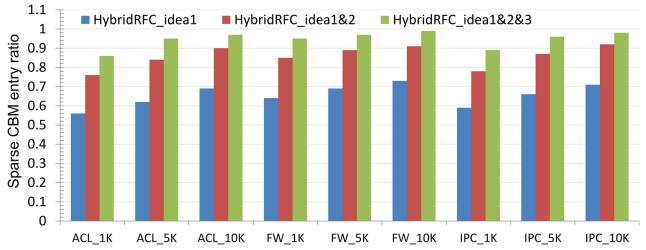

Fig. 12 *Ratio of sparse CBM entry in HybridRFC with different key ideas*

4.4 Worse-case memory access

Fig. 13 shows the performance in terms of the worst number of memory access for HybridRFC as well as HybridCuts and CutSplit, which are two latest algorithmic approach. Compared to HybridCuts and CutSplit, experimental results show that HybridRFC achieves an average of 3.5 × and 2.6 × improvement respectively, as well as 1.4 × and 1.9 × improvement with parallel implementation.

4.5 Throughput

Fig. 14 shows some simulation results on classification performance for HybridRFC as well as HybridCuts and CutSplit. We conduct simulations by classifying one million packets generated by ClassBench when it constructs the corresponding classifier. We measure the packet classification throughput with one-level caching which is called microflow in Open vSwitch. Experimental results show that HybridRFC runs faster than

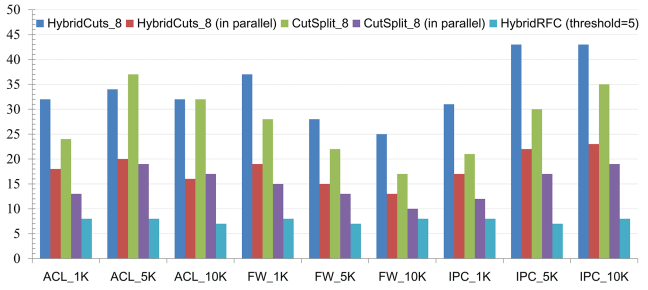

Fig. 13 *Worst-case memory access compared with the latest algorithmic approaches*

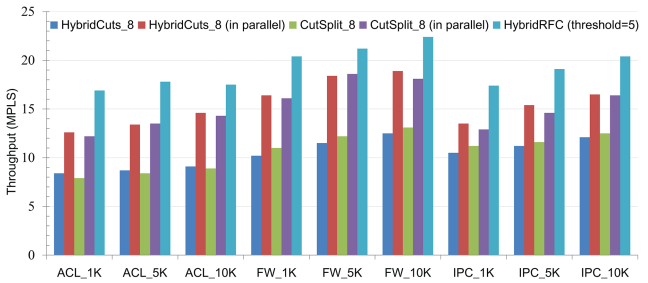

Fig. 14 *Throughput compared with the latest algorithmic approaches*

HybridCuts and CutSplit, achieves an average of 0.84 and 0.79 times speed-up respectively, as well as 0.24 and 0.27 times speed-up with parallel implementation.

5 Conclusion

Multi-field packet classification is not only an indispensable and challenging functionality of existing network devices, it also plays a central role in the forwarding plane of SDN. Despite almost two decades of research, algorithmic solutions still fall short of meeting the line-speed of high-performance network devices. Although decomposition based-schemes are well-suited for designing high-throughput network devices, most of them are still memory inefficient. In this paper, we propose an improved RFC algorithm called HybridRFC, which is a memory-efficient recursive scheme for fast multi-field packet classification. The main contributions of this work include: several novel observations on the sparseness of CBM tables and a refined framework with three key ideas to reduce memory consumption. By addressing the key problem of uncontrolled memory explosion of cross-product tables in RFC, HybridRFC can not only reduce the memory consumption to a practical level, but also improve pre-processing performance significantly.

6 Acknowledgments

This work is supported in part by the National Keystone R&D Program of China (nos. 2017YFB0803204 and 2016YFB0800101), by Shenzhen Peacock Project (no. KQJSCX2018032317474421), by the Natural Science Foundation of Chi- na (NSFC) (no. 61671001), by Guangdong Key Program (GD2016B030305005) and by Shenzhen Research Programs (JCYJ20170306092030521 and ZDSYS201703031405137).

7 References

[1] McKeown, N., Anderson, T., Balakrishnan, H., *et al.*: 'OpenFlow: enabling innovation in campus networks', *ACM SIGCOMM Comput. Commun. Rev.*, 2008, **38**, (2), pp. 69–74

[2] Liu, A.X., Meiners, C.R., Torng, E.: 'Packet classification using binary content addressable memory'. Proc. IEEE Int. Conf. on Computer Communications (INFOCOM), Toronto, Canada, April 2014, pp. 628–636

[3] Liu, A.X., Meiners, C.R., Torng, E.: 'TCAM razor: a systematic approach towards minimizing packet classifiers in TCAMs', *IEEE/ACM Trans. Netw.*, 2010, **18**, (2), pp. 490–500

[4] Liu, A.X., Meiners, C.R., Zhou, Y.: 'All-match based complete redundancy removal for packet classifiers in TCAMs'. Proc. IEEE Int. Conf. on Computer Communications (INFOCOM), Phoenix, USA, April 2008, pp. 111–115

[5] Rottenstreich, O., Tapolcai, J.: 'Lossy compression of packet classifiers'. Proc. ACM/IEEE Symp. on Architectures for Networking and Communications Systems (ANCS), Oakland, USA, May 2015, pp. 39–50
[6] Rottenstreich, O., Radan, M., Cassuto, Y., *et al.*: 'Compressing forwarding tables'. Proc. IEEE Int. Conf. on Computer Communications (INFOCOM), Turin, Italy, April 2013, pp. 1231–1239
[7] Che, H., Wang, Z., Zheng, K.: 'DRES: dynamic range encoding scheme for TCAM coprocessors', *IEEE Trans. Comput.*, 2008, **57**, (7), pp. 902–915
[8] Liu, H.: 'Efficient mapping of range classifier into ternary-CAM'. Proc. IEEE Annual Symp. on High-Performance Interconnects (Hot Interconnects), Stanford, USA, August 2002, pp. 95–100
[9] Rottenstreich, O., Cohen, R., Raz, D.: 'Exact worst-case TCAM rule expansion', *IEEE Trans. Comput.*, 2013, **62**, (6), pp. 1127–1140
[10] Rottenstreich, O., Keslassy, I.: 'On the code length of TCAM coding schemes'. Proc. IEEE Annual Symp. on Information Theory (ISIT), Texas, USA, June 2010, pp. 1908–1912
[11] Rottenstreich, O., Keslassy, I., Hassidim, A.: 'Optimal in/out TCAM encodings of ranges', *IEEE/ACM Trans. Netw.*, 2016, **24**, (1), pp. 555–568
[12] Li, W., Li, X., Li, H.: 'MEET-IP: memory and energy efficient TCAM-based IP lookup'. Proc. IEEE Int. Conf. on Computer Communication and Networks (ICCCN), Vancouver, Canada, August 2017, pp. 1–8
[13] Li, X., Lin, Y., Li, W.: 'GreenTCAM: a memory- and energy-efficient TCAM-based packet classification'. Proc. IEEE Int. Conf. on Computing, Networking and Communications (ICNC), Kauai, USA, February 2016, pp. 1–6
[14] Ma, Y., Banerjee, S.: 'A smart pre-classifier to reduce power consumption of TCAMs for multi-dimensional packet classification'. Proc. ACM Int. Conf. on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), Helsinki, Finland, August 2012, pp. 335–346
[15] Ruan, Z., Li, X., Li, W.: 'An energy-efficient TCAM-based packet classification with decision-tree mapping'. Proc. IEEE Region 10 Conf. (TENCON), Xian, China, October 2013, pp. 1–5
[16] Daly, J., Torng, E.: 'TupleMerge: building online packet classifiers by omitting bits'. Proc. IEEE Int. Conf. on Computer Communication and Networks (ICCCN), Vancouver, Canada, August 2017, pp. 1–10
[17] Yang, T., Liu, A.X., Shen, Y., *et al.*: 'Fast openflow table lookup with fast update'. Proc. IEEE Int. Conf. on Computer Communications (INFOCOM), Honolulu, USA, April 2018, pp. 2636–2644
[18] Daly, J., Torng, E.: 'ByteCuts: fast packet classification by interior bit extraction'. Proc. IEEE Int. Conf. on Computer Communications (INFOCOM), Honolulu, USA, April 2018, pp. 2654–2662
[19] Fong, J., Wang, X., Qi, Y.: 'ParaSplit: a scalable architecture on FPGA for terabit packet classification'. Proc. IEEE Annual Symp. on High-Performance Interconnects (Hot Interconnects), Santa Clara, USA, August 2012, pp. 1–8
[20] He, P., Xie, G., Salamatian, K., *et al.*: 'Meta-algorithms for software based packet classification'. Proc. IEEE Int. Conf. on Network Protocols (ICNP), Raleigh, USA, October 2014, pp. 308–319
[21] Li, W., Li, X., Li, H., *et al.*: 'CutSplit: a decision-tree combining cutting and splitting for scalable packet classification'. Proc. IEEE Int. Conf. on Computer Communications (INFOCOM), Honolulu, USA, April 2018, pp. 2645–2653
[22] Li, W., Li, X.: 'HybridCuts: a scheme combining decomposition and cutting for packet classification'. Proc. IEEE Annual Symp. on High-Performance Interconnects (Hot Interconnects), Santa Clara, USA, August 2013, pp. 41–48
[23] Li, X., Lin, Y.: 'TaPaC: a TCAM-assisted algorithmic packet classification with bounded worst-case performance'. Proc. IEEE Global Communications Conf. (GLOBECOM), Washington, USA, December 2016, pp. 1–6
[24] Qi, Y., Xu, L., Yang, B., *et al.*: 'Packet classification algorithms: from theory to practice'. Proc. IEEE Int. Conf. on Computer Communications (INFOCOM), Rio de Janeiro, Brazil, April 2009, pp. 648–656
[25] Vamanan, B., Voskuilen, G., Vijaykumar, T.N.: 'EffiCuts: optimizing packet classification for memory and throughput'. Proc. ACM Int. Conf. on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), New Delhi, India, September 2010, pp. 207–218
[26] Yang, T., Xie, G., Liu, A.X.: 'Constant IP lookup with FIB explosion', *IEEE/ACM Trans. Netw.*, 2018, **26**, (4), pp. 1821–1836
[27] Pak, W., Bahk, S.: 'FRFC: fast table building algorithm for recursive flow classification', *IEEE Communication Letters*, 2010, **14**, (12), pp. 1082–1084
[28] Tayor, D.E., Turner, J.S.: 'Scalable packet classification using distributed crossproducting of field labels'. Proc. IEEE Int. Conf. on Computer Communications (INFOCOM), Miami, USA, March 2005, pp. 269–280
[29] Xu, B., Jiang, D., Li, J.: 'HSM: a fast packet classification algorithm'. Proc. IEEE Int. Conf. on HSM: a Fast Packet Classification Algorithm (AINA), Taipei, Taiwan, March 2005, pp. 987–992
[30] Srinivasan, V., Varghese, G., Suri, S., *et al.*: 'Fast and scalable layer four switching'. Proc. ACM Int. Conf. on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), British Columbia, Canada, September 1998, pp. 191–202
[31] Gupta, P., McKeown, N.: 'Packet classification on multiple fields'. Proc. ACM Int. Conf. on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), Cambridge, USA, September 1999, pp. 147–160
[32] Tayor, D.E., Turner, J.S.: 'Classbench: a packet classification benchmark'. Proc. IEEE Int. Conf. on Computer Communications (INFOCOM), Miami, USA, March 2005, pp. 2068–2079