



MEBV: Resource Optimization for Packet Classification Based on Mapping Encoding Bit Vectors

Feng Guo^{1,2}, Ning Zhang¹(✉), Qian Zou^{1,2}, Qingshan Kong¹, Zhiqiang Lv^{1,2},
and Weiqing Huang^{1,2}

¹ Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
zhangning@iie.ac.cn

² School of Cyber Security, University of Chinese Academy of Sciences,
Beijing, China

Abstract. Packet classification plays a key role in network security systems such as firewalls and QoS. The so-called packet classification is to classify packets into different categories according to a set of predefined rules. When the traditional classification algorithm is implemented based on FPGA, memory resources are wasted in storing a large number of identical rule subfields, redundant length subfields, and useless wildcards in the rules. At the same time, due to the rough processing of range matching, the rules are extended. These problems seriously waste memory resources and pose a huge challenge to FPGAs with limited hardware resources. Therefore, a field mapping encoding bit vector (MEBV) scheme is proposed, which consists of a field-splitting-recombination architecture that can accurately divide each field into four mapping preparation fields according to the matching method, field reuse rate, and wildcard ratio, and also consists of four mapping encoding algorithms to complete the length compression of the rules, to achieve the purpose of saving resources. Experimental results show that for the 1K OpenFlow 1.0 ruleset, the algorithm can achieve a significant reduction in memory resources while maintaining high throughput and support range matching, and the scheme method can save an average of 38% in memory consumption.

Keywords: Packet classification · Bit-vector · Memory compression · Mapping encoding · FPGA

1 Introduction

With the rapid development of Internet technology and the gradual increase of network security requirements, people are looking for various solutions to cope with various network attacks while the business requirements are gradually

Supported by the Chinese Academy of Sciences Project under grant NO. KGFZD-145-21-03.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
L. Wang et al. (Eds.): WASA 2022, LNCS 13473, pp. 84–95, 2022.
https://doi.org/10.1007/978-3-031-19211-1_7

increasing. Packet classification is a technology that divides traffic into different flows based on a set of predefined rules. It can not only realize the blacklist function of separating and discarding the flow containing malicious traffic, and transparently transmitting other flows; it can also realize a similar whitelist function. As such, it is more challenging, especially in environments where packets must be processed at wire speed. And packet classification is also the core issue of OpenFlow-based software-defined networking (SDN) [1]. The ever-increasing number of fields and ever-expanding rulesets pose great challenges to practical packet classification solutions with high throughput and low memory consumption. From a hardware perspective, the main challenges for packet classification include: (1) supporting large rulesets, and (2) maintaining high performance.

In the past decade or so, software-based decision tree algorithms [2] and tuple space search algorithms [3] proposed for classical problems have been used on CPU processing platforms. However, the performance of software-based methods is limited by the CPU memory system. Hardware-based ternary content-addressable memory (TCAM) [4] solutions have also been widely used in industry due to their parallel lookup of wire-speed classification rules. But it has the disadvantages of being expensive, power-hungry, and limited capacity. Field Programmable Gate Arrays (FPGA) [5] have been widely used to process real-time networks. Algorithms based on bit vectors can make good use of the hardware parallelism of FPGAs through rule decomposition. The currently proposed bit-vector-based algorithm [6–14] can achieve high throughput by exploiting a homogeneous pipeline structure (PE) consisting of sorting processes. But as match fields and rulesets increase, the FPGA’s master clock frequency decreases rapidly and range matching is not supported. To solve the above issues, [9] divides the N-bit vector into smaller sub-vectors to improve the overall performance of classified PE in FPGA. And [12] proposed an algorithm to support range matching using precoding. However, the above approach does not reduce the required memory resources. Therefore, to achieve resource optimization while taking into account range matching and throughput, a memory-optimized scheme called MEBV based on field mapping encoding is proposed. Our contributions to this work include:

A field-splitting-recombination architecture: The architecture can accurately divide each field into four mapping preparation fields according to the matching method, field reuse rate, and wildcard ratio.

Four mapping encoding algorithm frameworks, HRME, PMME, WMME, RMME: According to the characteristics of the mapping preparation field, four different algorithms are used for mapping encoding to generate corresponding mapping vectors.

Superior memory optimization: Detailed performance evaluation of our proposed architecture on state-of-the-art FPGAs. We show in post-place-and-route results that for a 1K 12-tuple ruleset, the architecture can save 38% memory consumption on average.

Higher throughput: Compared with algorithms that support range matching, this scheme saves resources while keeping the impact on throughput within 5%, and meets the requirements for wire-speed packet classification.

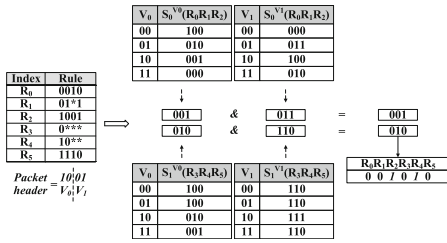


Fig. 1. Schematic diagram of stride BV algorithm. ($W = 4, s = 2$)

Field	Abbreviation	Bits
ingress port	IN_PORT	16
ethernet source address	SMAC	48
ethernet destination address	DMAC	48
Ethernet type	ETH_TYPE	16
VLAN ID	VLAN_ID	12
VLAN priority	VLAN_PRI	3
source IP	SIP	32
destination IP	DIP	32
IP protocol	PROTOCOL	8
IP ToS bits	IP_ToS	6
transmission source Port/ICMP Type	SPORT	16
transmission destination Port/ICMP Code	DPORT	16
total # of bits		253

Fig. 2. Field length and abbreviated name of OpenFlow entry.

2 Related Work

2.1 BV-Based Packet Classification

The Stride BV [7] algorithm divides a field with a bit width of W bits into $m = W/s$ subfields, where s is the length of a subfield, and the divided subrules are encoded and stored separately. This method reduces the bit width of the internal signal from W to s , which improves the throughput, but consumes more memory and is not suitable for range matching, the algorithm is shown in Fig. 1, it illustrates a Stride bv-based packet classification method. The bit vector $S_i^{V_j}$ is used to represent the matching result of V_j to the matching *subfield* _{j} corresponding to the *subruleset* _{i} . In this example, s is set to 2 and n is set to 3. For example, in Fig. 1, if the input packet header has $V_0 = 10$ in *subfield*₀ of *subruleset*₁, extract $S_1^{V_0} = 010$; this means that only rule R_4 of subruleset 1 matches the input in that subfield. The FSBV [8] algorithm is a special case, $s = 1$. When the bit width W increases, the number of pipeline stages will increase linearly with W , which will also cause a large delay in packet processing. The subsequently proposed two-dimensional pipelined stride bit vector (2D Stride BV, hereinafter referred to as 2D BV) [9], which is based on the Stride BV [7] algorithm, supports dynamic updates and further improves throughput. But the above problem still exists. Therefore, in response to the problem of memory consumption, the wildcard-removed bit-vector (WeeBV) [10] deletes the address space storing wildcards as much as possible to save memory by making full use of the characteristics of the ruleset. For the range matching problem, range bit vector encoding (RBVE) [12] has the same characteristics as Stride BV [7], and uses specially designed code to store the precomputed results in memory, which can solve the range matching problem. The subsequently proposed Range Supported Bit Vector (RSBV) [13] further improves RBVE and achieves high processing speed. However, the above methods are one of the existing problems, and cannot support range matching while saving memory.

2.2 Motivation

The process of packet classification: Given a ruleset, extracting the header field of the packet when the packet is input. Matching the header field with the corresponding fields of the ruleset, and processing the packet according to the specified action in the matched result.

The most classic is to use 5-tuple for packet classification, examining each incoming packet for the following header tuples: Source IP, Destination IP, Source Port, Destination Port, Protocol. But to accommodate today's security policies, a simple five-tuple is not enough. The OpenFlow 1.0 [15] header includes 12 fields, and the detailed description of the attributes of each field is shown in Fig. 2. By analyzing the characteristics of the OpenFlow 1.0 ruleset in [10], it is found that there are various types of search methods for different fields, and many fields have problems such as single value and a high proportion of wildcard characters: For example, the IP address field requires prefix matching, the port field requires range matching, the Ethernet type field has a fixed value and a large bit width, and the wildcard ratio of fields such as IP_ToS is high. Since the meaning of wildcards is whether the bit matches whether the bit is 0 or 1, these wildcards don't mean anything.

3 Proposed Scheme

3.1 Mapping Encoding Bit Vector (MEBV) SCHEME

MEBV is to decompose the entry into the various subfields in Fig. 3. Based on the matching form of each subfield or the number of wildcards, the F subfields are recombined into 4 mapping preparation fields. Then, different mapping algorithms are applied to each mapping preparation field for encoding to generate mapping vectors. Assuming that the lengths of the mapping preparation fields are L_1, L_2, L_3, L_4 , and the lengths of the mapping encoding fields are D_1, D_2, D_3, D_4 . Respectively, there will be $L_1 + L_2 + L_3 + L_4 \gg D_1 + D_2 + D_3 + D_4$.

For example, the OpenFlow 1.0 [15] header includes 12 fields that can be split into 12 subfields. Then analyze the characteristics of each subfield: the reuse rate of the Ethernet type and IP PROTOCOL fields is very high. Source IP and destination IP are mostly prefix matching, and the multiplexing rate is high. Source Port and Destination Port are mostly range matching. The remaining fields are exact or wildcard matches. According to the wildcard ratio of OpenFlow ruleset 12-tuple organized in MsBV [11], the fields can be divided more clearly: these fields can be arranged according to the wildcard ratio from small to large, to complete the subsequent operation of deleting wildcards. Through analysis, the detailed subfield division and mapping preparation field classification results after applying the field rearrangement technology are shown in Fig. 3.

According to the specific matching form of the field (prefix matching, range matching, exact matching, wildcard matching, etc.) or specific characteristics, the mapping preparation field adopts a specific algorithm to compress the length

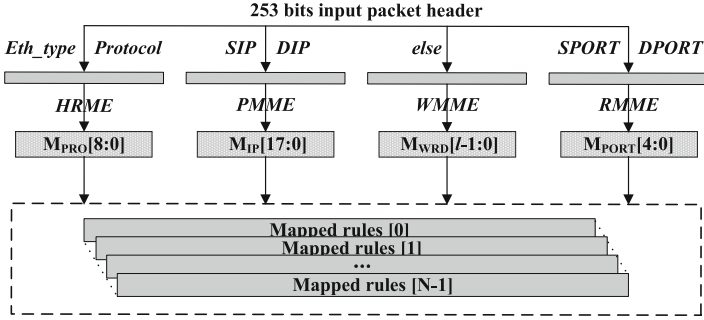


Fig. 3. MEBV scheme.

of the rule and improve the matching efficiency. The four specific algorithms of MEBV are: HRME: high reuse field mapping encoding, PMME: prefix matching field mapping encoding, WMME: wildcard matrix field mapping encoding, and RMME: range matching field mapping encoding.

Next, each mapping encoding algorithm will be introduced in detail. Please note that when this solution is applied to packet classification, rules can be customized according to the application scenario, which can achieve good scalability while compressing resources.

3.2 HRME: High Reuse Field Mapping Encoding

By analyzing a large number of rulesets and observing real traffic data, there is a serious waste of resources in the highly multiplexed field composed of the Ethernet type field and the IP protocol type field. The EtherType field has 2 bytes and 16 bits. If the Stride BV [7] algorithm is applied and divided into steps of 8, $2 * 2^8 = 512$ values are also required.

However, most of the values for this field are clustered around 0x0800 (IPv4), 0x0806 (ARP), 0x8100 (IEEE 802.1Q frame label), 0x86DD (IPv6) and wildcards, etc., while for our packet classification techniques such as OpenFlow1.0 ruleset, all belong to Ethernet II and IEEE802.3 frames. Therefore, the value of this field is more fixed. To support the blacklist function or the whitelist function, a linear mapping method can be applied to this field to save a lot of resources. The linear mapping results are shown in Table 1. The 5-bit width is selected here mainly to consider the scalability of the coding results of this field. If the application network environment is more complex, it can be expanded on this basis.

As shown above, only the five commonly used values and wildcards need to be encoded, and the other rarely used values are classified together. The method will not fail when doing a match, since this is only one of the fields. To get the correct result, the scheme needs to get the matching result of all fields.

The IP protocol type field is very similar to the Ethernet type field. The commonly used values of the IP protocol type field are 0x01 (ICMP), 0x06

Table 1. EtherType field mapping encoding result.

ETH_TYPE	V_{ETH}	M_{ETH}
IPv4	0x0800	00000
ARP	0x0806	00001
802.1Q tag	0x8100	00010
IPv6	0x86DD	00011
MPLS	0x8847	00100
else		11111

Table 2. IP protocol type field mapping encoding result.

IP_PROTOCOL	V_{PRO}	M_{PRO}
ICMP	0x01	0000
IGMP	0x02	0001
TCP	0x06	0010
UDP	0x17	0011
ESP	0x50	0100
else		1111

(TCP), 0x17 (UDP), etc., so the linear mapping method also can be used to get the encoding result, as shown in the Table 2.

3.3 PMME: Prefix Matching Field Mapping Encoding

In a set of rulesets, the consumption of redundant space can be effectively reduced by grouping DIP and SIP with the same prefix length. However, if linear classification is used, when the number of ruleset entries N increases, the space used for classification will explode, and the classification work will be extremely cumbersome. So in this scheme decided to use nonlinear classification, such as hash algorithm. By hash mapping the destination IP (or source IP) fields classified by prefix length DPL_i (or SPL_i) ($i = 0, 1, 2, \dots, 32$), a hash value of a specific length H is generated and stored in the corresponding memory. The architecture is as follows:

The ruleset has a total of N rules, which are classified according to the prefix length DPL_i ($i = 0, 1, 2, \dots, 32$). When $DPL_i < H$, the corresponding hash values are stored in the same memory, and the memory size is $n * 2^H$; when $DPL_i > H$, each P_i will maintain a piece of memory for storing the hash value generated by the IP address belonging to its own prefix length, and the memory size is also $n * 2^H$, where n represents the number of rules stored in a RAM. The processing method of SIP is the same as above. The following Fig. 4 illustrates the mapping and encoding process of the prefix matching field.

3.4 WMME: Wildcard Matrix Field Mapping Encoding

For wildcard matrix fields, the encoding of the field mapping will be different from the above. Because wildcards represent any value [11], that is, whether the field value is "0" or "1", it will be matched. So can aggregate fields with a large number of wildcards together by field rearrangement to form a wildcard matrix. When a match operation is performed, the matching result output by this matrix is 1 by default, that is, full matching. Therefore, eliminating a large number of useless wildcards in memory is also an effective means to reduce the waste of resources. A schematic diagram of applying field rearrangement techniques and rule rearrangement techniques to all rules to form a matrix of wildcards is shown in Fig. 5.

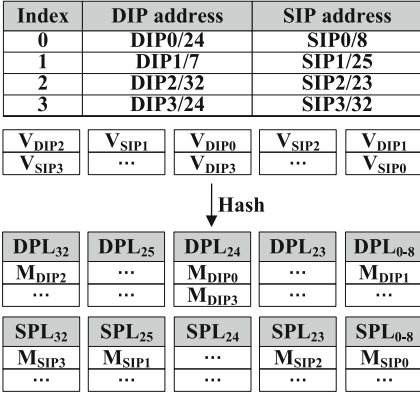


Fig. 4. Process of PMME.

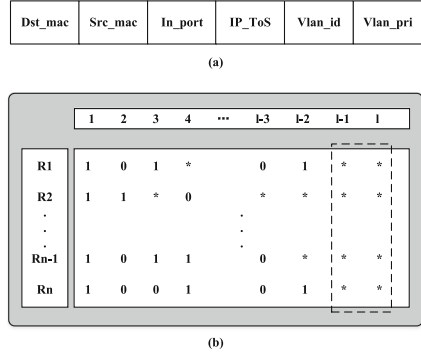


Fig. 5. Process of WMME.

3.5 RMME: Range Matching Field Mapping Encoding

The source port and destination port fields in a ruleset are usually 16-bit ranges. A 16-bit range is represented by $[LR, UR]$, where LR is the lower range limit and UR is the upper range limit. In this section, a Range Bit Vector Encoding (RBVE) [12]-like the scheme is used. In this scheme, a 16-bit range is divided into two 8-bit subranges. When performing rule matching, this classification method will cause the matching results of the latter sub-range to be related to the previous. Therefore, to complete the rule matching operation, a two-stage pipeline can be designed to place the two sub-range fields. Let V_{PORT} be a 16-bit input address and divide A into two sub-range fields in stride of 8, $V_{PORTi}, i = 0, 1$. The specific implementation process used to generate the matching signal in the RBVE [12] algorithm is shown in the following pseudocode **Algorithm 1,2** in Fig. 6. The range matching field mapping encoding process has been introduced, and the matching problem and operations will be discussed in the following chapters.

3.6 Packet Rule Matching

To better demonstrate our rule matching process, an example is as follows in Fig. 7, it shows an example of applying the MEBV algorithm to build a mapped ruleset and matching all fields of the packet to it.

We map and encode the 4 rules in the ruleset to generate mapping rules and store them in memory. When a data packet is input, the corresponding mapping encoding will be performed first to obtain the mapping result as shown in the figure. Since $s = 9$, the mapping vector produced by the eth type and protocol fields can be combined into a stride as the input address. The hash result of the source IP and the destination IP is also nine digits, which is also the length of a stride. Subsequent wildcard fields will also be divided in steps of s .

Algorithm : RMME.		
M_{PORT0} : 111 if $LR_0 < V_{PORT0} < UR_0$; 100 if $LR_0 < UR_0 \ \& \ V_{PORT0} = UR_0$; 010 if $LR_0 = V_{PORT0} = UR_0$; 001 if $LR_0 < UR_0 \ \& \ V_{PORT0} = LR_0$;	M_{PORT1} : 10 if $V_{PORT1} \leq UR_1$; 01 if $V_{PORT1} > LR_1$;	Match : 1 if $(M_{PORT0} = 001 \ \& \ M_{PORT1} = 01) \ \ (M_{PORT0} = 100 \ \& \ M_{PORT1} = 10) \ \ ((M_{PORT0} = 001 \ \ 010 \ \ 100) \ \& \ M_{PORT1} = 11)$

Fig. 6. Process of RMME.

When inputting a data packet, the 9-bit mapping vector of the eth type and the protocol field is used as the first set of input, all possible 9-bit mapping vectors of the destination IP are used as the second set of input, and all possible 9-bit mapping vectors of the source IP are used as the third set of input. This example is a brief introduction, so except for the fields above, the rest of the rule's fields are set to wildcards. Here, the rules are sorted by wildcard ratio, and fields with high wildcard ratios end up in a wildcard matrix. Fields with a low wildcard ratio are listed first and filled into memory according to the actual value, without mapping and encoding.

The port field is range matching, so this field adopts the RBVE algorithm. The mapping code value of this field will be directly used for matching judgment, and the composition of the mapping rule does not include this field.

3.7 Hash Collision Resolution

There will always be a problem with hash collisions when using hashing algorithms. The solution to the hash conflict in this scheme is to maintain two sets of hash algorithms. When the result of the first hash calculation produces a hash conflict, the second set of hash algorithms will be enabled. If the calculation results still conflict, the IP data will be temporarily stored, and the write address counter in this memory will be read after all the prefix length rules have been configured. Then choose the lowest address with a write address counter of 0, and force the hash result to encode this address value and store it in memory. At the same time, to minimize hash collisions under normal circumstances, the bit width of the hash value will be increased as much as possible.

4 Experimental Results and Analysis

In this section, we present the experimental setup and experimental results, which are measured in terms of space complexity, throughput, and resource consumption.

Synthetic classifiers: To test the performance of our scheme and existing techniques, we used ClassBench-ng [16], an excellent tool inherited from ClassBench [17], to generate OpenFlow1.0 rules. ClassBench-ng [16] provides torrents from real-life rules to get performance as close to practice as possible.

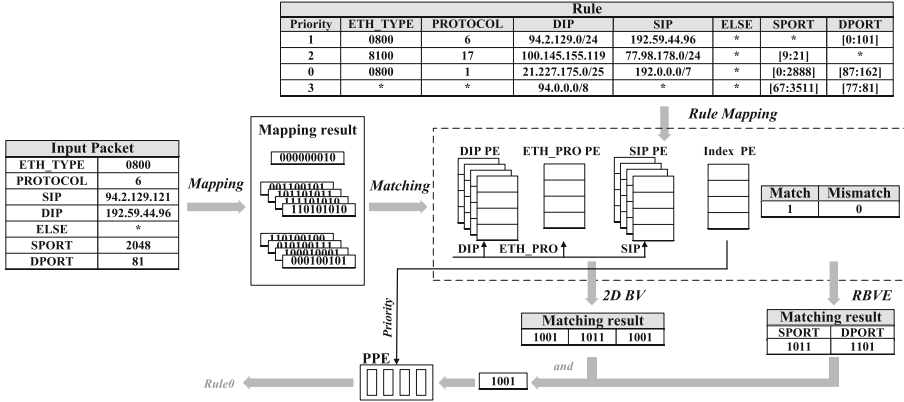


Fig. 7. MEBV: mapping encoding of rules and rules matching of packets.

The rulesets such as Accesses Control List (ACL), Firewall (FW) and IP Chain (IPC) generated by ClassBench [17] only contain traditional 5-tuple, so we use OpenFlow1.0 containing 13-tuple to prove the advanced nature of the algorithm. If excellent performance can be achieved on the OpenFlow 1.0 ruleset, so on other rulesets.

Implementation platform: The experimental environment is a Xilinx Virtex7 xc7vx690t [18] FPGA device. Limited to the experimental platform, simulation software is used to test the performance: Vivado 2018.3 and Modelsim 10.4, the results presented here are based on the post-synthesis and route performance.

4.1 Space Complexity

Assuming that there are N rules in the ruleset, the rule bit width is W , the field is divided by stride (s), and each memory stores n rules, then the space complexity of MEBV is calculated as follows. There are a total of F fields in the rule, of which f_0 fields need to complete HRME, f_1 fields need to complete PMME, f_2 fields need to complete WMME, and f_3 fields need to complete RMME. In the mapping preparation stage, f_0 and f_1 generate 8-bit mapping vector, f_2 generates l -bit mapping vector, and f_3 generates 3-bit mapping vector and 2-bit mapping vector in two stages. Therefore, the theoretical space complexity of MEBV is given by the following equation [13]

$$S_{Theory}(R) = S_{ETH_PRO}(R) + S_{IP}(R) + S_{WRD}(R) + S_{PORT}(R) \quad (1)$$

$$= (9 \times (f_0 + f_1) \times 2^9 + ((\frac{l}{s}) \times f_2 + 5 \times f_3) \times 2^8) \times N \quad (2)$$

Because the memory resources that can implement parallel operations are limited. The minimum granularity of Xilinx FPGA's [18] 18Kb and 36Kb Block RAM (BRAM) primitives is 36bit×512 and 72bit×512. Therefore, we set $s=9$,

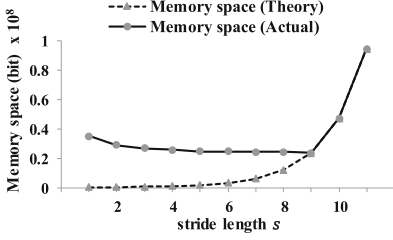


Fig. 8. Simulation of space complexity.

Algorithm	Space Complexity	Result	Range
TCAM ^[2]	$O(2(W-1)^F \times N)$	2.8×10^{35}	yes
FSBV ^[6]	$O(W \times F \times N)$	3.1×10^6	no
Stride BV ^[15]	$O((W/s) \times F \times N \times 2^2)$	9.9×10^7	no
2D BV ^[7]	$O((W/s) \times F \times N \times 2^2)$	9.9×10^7	no
RBVE ^[10]	$O((4(W/s) - 3) \times F \times N \times 2^2)$	3.9×10^8	yes
RSBV ^[11]	$O((4(W/s) - 3) \times F \times N \times 2^2)$	3.9×10^8	yes
WeeBV ^[8]	$O(((W - l_w)/s) \times F \times N \times 2^2)$	4.6×10^7	no
MEBV	$O(S_{actual}(R))$	1.6×10^7	yes

Fig. 9. Space complexity comparison.

then the memory depth required for each stage is $d=512$, and there is $F = f_0 + f_1 + f_2 + f_3$, then the formulate Summarized as

$$S_{actual}(R) = (9 \times (f_0 + f_1) + (\frac{l}{s}) \times f_2 + 5 \times f_3) \times N \times d \quad (3)$$

According to the experimental ruleset, $W = 253$, $N = 1024$, $F = 12$, $f_0 = 2$, $f_1 = 2$, $f_2 = 6$, $f_3 = 2$, $l = 25$. The value of l here is obtained from [11]. In addition, according to encoding rules, f_0 and f_2 should be 1, when calculating. In summary, the space complexity as a function of s is shown in Fig. 8. It is found that the optimal step size s in the FPGA implementation is 9, so it is decided to use a step size of 9 in 2D BV [9]. In the parallel architecture RBVE [12], since the length of the port field is 16 bits, the step size is 8.

When $s = 9$, compared with other algorithms, the space complexity is shown in Fig. 9, and it is marked whether to support range matching. Where l_w represents how many bits of wildcards are in the WeeBV [10] algorithm.

4.2 Throughput

Next, the scheme compared the throughput of multi-field packet classification. When there are $N = 1K$ rules in the ruleset, the rule bit width is $W = 253$, the $s = 9$ of 2D BV [9], the $s = 8$ of RBVE [12], and $l = 36$ rules are stored in each memory. The simulation results show that the maximum clock frequency of the MEBV algorithm is as high as 127.15 MHz. If the block RAM used in the MEBV algorithm is set as a real dual-port RAM, the throughput of the algorithm can reach 254.30 MPPS. The comparison of the MEBV algorithm with existing work is shown in Fig. 10. It can be found that the algorithm can achieve higher throughput while reducing resource consumption and effectively supporting range matching. However, the throughput of the MEBV algorithm is 14% lower than 2D BV and 1.2% lower than RBVE. The main reason is that the algorithm is executed in parallel by 2D BV and RBVE, and the throughput is determined by the smallest channel. Furthermore, due to the mapping encoding of this algorithm, this will lead to a certain decrease in the clock frequency.

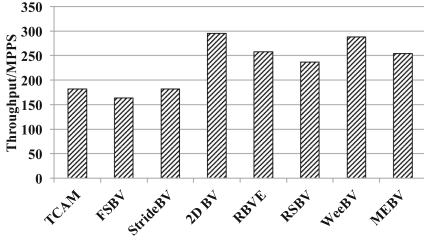


Fig. 10. Throughput comparison between algorithms.

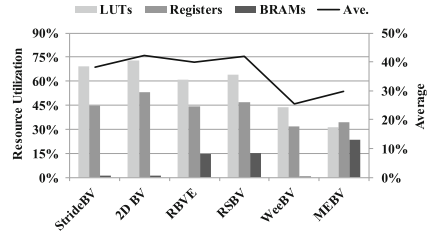


Fig. 11. Comparison of resource utilization between algorithms.

4.3 Resource Consumption

Then, we mainly focus on the FPGA resource consumption of the proposed MEBV algorithm. $N = 1024, W = 253, 2D\ BV$ [9] for $s = 9$, $RBVE$ [12] for $s = 8$, $l = 36$. The FPGA resource consumption comparison of different flow classification algorithms is shown in Fig. 11. The MEBV uses mapping encoding to reduce the length of storage rules, which can save a lot of register resources and LUT resources, but because it supports range matching and the results of mapping encoding are stored in BRAM, the consumption of BRAM resources increases. But in general, a large number of register resources are often built-in FPGA chips (Virtex 7 xc7vx690t FPGA [18] has built-in 52Mb BRAM resources). In this scheme, the BRAM resource consumption rate is 23.3%, while the LUT resource consumption rate is 43.8%, so the extra BRAM cost will not become a bottleneck, and the saved resources can support more strategies. To make the experimental results more representative and comparative, the resource utilization here is the result after placement and routing.

The broken line in the figure represents the average resource utilization, which can better reflect the superiority of our scheme. It is worth noting that the algorithm has the smallest average resource consumption among the algorithms that support range matching. That is to say, under the premise of the same resources, the scheme can support more strategies, which undoubtedly further reduces the bottleneck caused by resources.

5 Conclusion

In this paper, we propose MEBV, a memory-optimized scheme based on field mapping encoding bit-vectors, which achieves resource optimization while considering range matching and throughput. Our proposed solution can save 32.6% of LUT resources and 12.4% of register resources compared to state-of-the-art algorithms [13] that support range matching, with throughput impact remaining within 2%. Compared with the matching algorithm [9], 41.6% of LUT resources and 18.4% of register resources can be saved, and the impact of throughput remains within 12%. Meet the requirements of wire-speed packet classification.

References

1. McKeown, N., et al.: OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Comput. Commun. Rev.* **38**(2), 69–74 (2008)
2. Erdem, O., Bazlamaççi, C.F.: Array design for Trie-based IP lookup. *IEEE Commun. Lett.* **14**(8), 773–775 (2010)
3. Song, H., Turner, J., Dharmapurikar, S.: Packet classification using coarse-grained tuple spaces. In: 2006 Symposium on Architecture For Networking And Communications Systems, pp. 41–50. IEEE (2006)
4. Yu, F., Katz, R.H., Lakshman, T.: Efficient multimatch packet classification and lookup with TCAM. *IEEE Micro* **25**(1), 50–59 (2005)
5. Fu, W., Li, T., Sun, Z.: FAS: using FPGA to accelerate and secure SDN software switches. *Secur. Commun. Netw.* **2018** (2018)
6. Lakshman, T., Stiliadis, D.: High-speed policy-based packet forwarding using efficient multi-dimensional range matching. *ACM SIGCOMM Comput. Commun. Rev.* **28**(4), 203–214 (1998)
7. Ganegedara, T., Prasanna, V.K.: StrideBV: single chip 400G+ packet classification. In: 2012 IEEE 13th International Conference on High Performance Switching and Routing, pp. 1–6. IEEE (2012)
8. Jiang, W., Prasanna, V.K.: Field-split parallel architecture for high performance multi-match packet classification using FPGAS. In: Proceedings of the Twenty-First Annual Symposium on Parallelism in Algorithms and Architectures, pp. 188–196 (2009)
9. Qu, Y.R., Prasanna, V.K.: High-performance and dynamically updatable packet classification engine on FPGA. *IEEE Trans. Parallel Distrib. Syst.* **27**(1), 197–209 (2015)
10. Li, C., Li, T., Li, J., Li, D., Yang, H., Wang, B.: Memory optimization for bit-vector-based packet classification on FPGA. *Electronics* **8**(10), 1159 (2019)
11. Shi, Z., Yang, H., Li, J., Li, C., Li, T., Wang, B.: MsBV: a memory compression scheme for bit-vector-based classification lookup tables. *IEEE Access* **8**, 38 673–38 681 (2020)
12. Chang, Y.-K., Hsueh, C.-S.: Range-enhanced packet classification design on FPGA. *IEEE Trans. Emerg. Top. Comput.* **4**(2), 214–224 (2015)
13. Zheng, L., Jiang, J., Pan, W., Liu, H.: High-performance and range-supported packet classification algorithm for network security systems in SDN. In: 2020 IEEE International Conference on Communications Workshops (ICC Workshops), pp. 1–6. IEEE (2020)
14. Zhou, Q., Yu, J., Li, D.: TSSBV: a conflict-free flow rule management algorithm in SDN switches. In: 2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring), pp. 1–5. IEEE (2021)
15. Heller, B.: OpenFlow switch specification, version 1.0. 0. Wire, December 2009
16. Matoušek, J., Antichi, G., Lučanský, A., Moore, A.W., Kořenek, J.: ClassBench: recasting ClassBench after a decade of network evolution. In: 2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), pp. 204–216. IEEE (2017)
17. Taylor, D.E., Turner, J.S.: ClassBench: a packet classification benchmark. *IEEE/ACM Trans. Network.* **15**(3), 499–511 (2007)
18. XA Programmable: Series FPGAS overview 7