A Multilayer Iteration-Based Improved Recursive Data Flow Matching Algorithm

Yanjing Caib,∗

1Jiangsu Vocational College of Business, Nantong, Jiangsu, China

*Corresponding author e-mail: caiyanjing0502@ntgx.edu.cn

Abstract. Differentiated service for packets entering the network is available through packet matching. Network security and differentiated services mean an inevitable choice for routers. Recursive data flow matching algorithm (RFC) is a high performance packet matching algorithm. However, with the increase of rule dimension and scale in the rule base, system memory consumption is unavoidable. This paper lowers memory consumption via improvement on RFC by dividing the rule base into several subsets and storing each rule in a separate subset. In addition, a variety of methods are used to streamline the RFC data structure for further improvement in algorithm speed and memory performance. The experimental results show that the improved algorithm of RFC greatly reduces the overall memory consumption of RFC, while greatly improving package matching performance.

Keywords: Packet Matching, RFC, Differentiated Service

1. Preface
Packet matching algorithm is to identify the data packets through the router, providing access control for the Internet to improve network service quality and offer differentiated services based on different customer requirements. Packet matching algorithm, The IP header field information generally involves source IP address, destination IP address, source port,. Each rule also has a behavior field for corresponding treatment of matching packets. Each rule generally has a priority.

Ternary content addressable memory (TCAM) is widely used in rule search. It, however, cannot directly deal with rule matching of range type, but need conversion of range rule into prefix rule type, which reduces space efficiency [1]. RFC is a well-known conditional high-performance algorithm requiring very low dimension and not big rule base scale. Otherwise, dimension disaster and scale disaster are possible.

2. The basic idea for improvement
This paper illustrates the necessity of rule base division in an example. Table 1 shows a rule base consisting of two header fields. There are five combinations in the source address (SA) field: 0 *(R3, R6), 010 *(R3, R4 R6), 1 *(R2, R6), 1100(R1, R2, R6) and 1110 (R2,R5 R6). The tag before brackets is the prefix matching of the rule in corresponding brackets. There are six combinations in the field of destination (DA): * (R5), 110 * (R5, R6), 1011 (R1, R5), 0 * (R4, R5), 010 * (R2, R4, R5) and 00 *
(R3, R5). Hence, the two header fields have 30 (= 6 * 5) entities after the outer product execution. The number of entities obtained by outer product can be reduced by dividing the rule base. Figure 1 illustrates the idea of rule division in a geometric method. These rules are divided into three subsets (R1 R5 R6), (R2, R5) and (R3, R4). The number of outer product entities for each subset is 9 (= 3 * 3), 4 (= 2 * 2), and 4 (= 2 * 2). Therefore, the total number of outer product entities is reduced to 17. Compared with the original, division of rule base can effectively reduce the number of outer product entities, thereby improving storage efficiency [2,3].

![Figure 1. A geometric description of rule in table 1](image)

The decision tree algorithm divides the rule base into subsets based on the rule's field attributes. When the attribute for rule base division is a field value, the decision tree provides rules of the same subset which are close to each other in geometric description. In this way, only one subset is accessed when the decision tree carries out packet matching. The number of subsets is controlled by adjusting the number of rules on one leaf node. However, the rule-based decision tree algorithm does not meet the second division requirement. As the wildcard specification is common in rule description of rule base, the geometric method for rule base division can only reduce, but not avoid rule repetition. There are several ways proposed to reduce duplicate rules also several algorithms use multiple decision trees to improve efficiency of the rule base division and other division algorithms divide rule set based on different attributes. None of these methods can totally avoid rule repetition within reasonable consumption [4,5].

We use an example to illustrate the above process. Table 2 shows the rule base composed of seventeen rules of five fields. Set the threshold to 4. In the first iteration of the division, source address field prefix has the largest different number, so it is chosen to divide the rule into three subsets, each of which corresponds to a node of the tree. As shown in Figure 2, the storage rule of the left child node of the root node has a source address field below the selected point, while the right child node of the root node stores a larger rule source address field. The middle root node stores the unclassified rules. Since the number of rules for the left and right child nodes exceeds 4 bits, further division is needed to generate a smaller subset. All unclassified rules in the decision tree (including R15, R16 R12, and R13) form the root nodes of the second decision tree. Since the size of the subset corresponding to the root node does not exceed the threshold, the second decision tree has only one node [6].
Figure 2. The decision tree

For an incoming packet, this complete search process first traverses the indexed RFC data structure to find the matching index rules, and then continues to search for a subset matching the index rules by accessing the corresponding RFC data structure. The algorithm framework proposed in this paper consists of six RFC data structures, five of which are subsets on the results and one about the index rule. Table 4 shows the outer product entities of the original RFC and the algorithm outer product entity presented in this paper. In this example, the algorithm proposed in this paper reduces outer product entity of the original RFC by 63%.

Table 1. The outer product table entity of original RFC and the algorithm presented in this paper at each phase

| RFC      | The proposed algorithm | All table entities |
|----------|------------------------|--------------------|
|          | Phase 1 | Phase 2 | Phase 3 | Phase 1 | Phase 2 | Phase 3 | RFC | The algorithm in this paper |
| Entity   | 1905    | 224     | 420     | Entity  | 665     | 133     | 155  | 2459 | 953     |

3. Further improvement

In this section, three techniques are proposed to further improve the time and memory consumption performance of the proposed algorithm.

A. Merge small subsets

Small subsets may be generated during rule base division. Such subsets will result in invalid RFC data structures and eventually extra memory access to these data structures. To avoid this, a threshold is designed, and when the number of rules for a subset is less than this threshold, merge it with other subsets. These merged subsets are stored in a common RFC data structure.

B. Merge different RFC data structures in the first phase

K + 1 RFC data structures should be designed to divide the rule base into K groups. Each RFC data structure needs to be traversed individually, so that 7 * (K + 1) memory access is required in the index array in the first phase to retrieve the corresponding eqID. The same fast index array despite different RFC data structures can be merged to reduce memory access times. In this way, one memory access can obtain the same fast eqIDs of different RFC data structures. The memory access times retrieved by RFC data structure in the first phase drop from 7 * (K + 1) to 7.

C. Reduce memory consumption
To store eqIDs in the first phase, each RFC data structure requires six 16-bit blocks and one 8-bit block. Each 16-bit block search table is an index array of 216 entities, while each 8-bit block search table is an index array of 28 entities. If the rule base is divided into K subsets, 6 \times 216 + 28 \times (k + 1) entities are required in the first phase. To reduce memory consumption, a binary search array can be designed in the first phase to replace the index array. For each index array, eqIDs stored in the adjacent entity can be the same. They can be merged into one interval, so that the first entity until the last entity shares the same eqID. In this way, the index array of 2 w entities can be converted into an n-interval array applicable to binary search. This method can reduce the memory consumption in the first phase [7].

4. Numerical experiment

The experimental results consist of three parts: the first part describes the balance between time and memory consumption of subsets of different sizes; the second part displays the performance improvement based on merging thresholds of different subsets; the final part is a study on performance comparison between the proposed algorithm and existing algorithms.

A. Performance balance under differently-sized subsets

The number of rules for a subset in the first part is determined by a factor. Define the factor d1, and then the rule size of the subset is equal to the total number of rules divided by d1. Divide the rule set until the number of rules in each subset is less than the threshold. Three factors are used in the experiment: 4, 8, 16, and the best performance factor will be selected in the next evaluation [8].

Figure 3 shows the memory requirements and memory access times in the worst case scenario under three differently-sized subsets of three different types of rule bases. As shown in Fig. 3 (a), as the number of rules in the subset decreases, that is, the number of subsets increases, the memory requirements gradually deteriorate. However, Figure 3 (b) also shows that as the number of rules in the subset increases, that is, the number of subsets decreases, memory access increases. This is because the incoming packet may set after trade-offs the factor d1 to 8, a value that can better balance storage and speed performance.

![Figure 3](image)

**Figure 3.** (a) Memory performance of the three types of rule bases under the three factors
5. Conclusion
This paper presents an important algorithm based on RFC's improvement. RFC, though a high-performance packet matching algorithm, consumes a lot of memory when generating the outer product table, and thus unsuitable for large-scale rule bases. To improve the memory performance, this paper proposes an improved algorithm to divide the rule base into several subsets. Rules in a subset are stored in a RFC data structure, and each subset is described by an index rule. All index rules are stored in a RFC index that points to the corresponding RFC data structure. Further, three techniques are proposed to improve memory and speed performance. Three rule bases are used to evaluate the algorithm performance in this paper. The numerical experiments show that the proposed algorithm has the best comprehensive performance.

Acknowledgments
Nantong City Science and Technology Project (No.: MSZ20156) funding.

References
[1] L. Choi, H. Kim, S. Kim, and M. H. Kim, "Scalable Packet Classification Through Rulebase Partitioning Using the Maximum Entropy Hashing," IEEE/ACM Trans. Netw., vol. 17, no. 6, pp. 1926-1935, 2009.
[2] P. Fong and T. Nian-Feng, "HaRP: Rapid Packet Classification via Hashing Round-Down Prefixes," IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 7, pp. 1105-1119, 2011.
[3] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," in Proceedings of Hot Interconnects VII, pp. 34-41, 1999.
[4] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in Proceedings of ACM SIGCOMM, pp. 213-224, 2003.
[5] B. Vamanan, G. Voskuilen, and T. N. Vijaykumar, "EffiCuts: Optimizing Packet Classification for Memory and Throughput," SIGCOMM Comput. Commun. Rev., vol. 41, no. 4, pp. 207-218, 2010.
[6] Http://www.arl.wustl.edu/~hs1/PClassEval.html
[7] X. Bo, J. Dongyi, and L. Jun, "HSM: A Fast Packet Classification Algorithm," in Proceedings
of AINA 2005, pp. 987-992, 2005.

[8] X. Sun, S. K. Sahni, and Y. Q. Zhao, "Packet Classification Consuming Small Amount of Memory," IEEE/ACM Trans. Netw., vol. 13, no. 5, pp. 1135-1145, 2005.