Memory Optimizations for Packet Classification Algorithms in FPGA

Viktor Puš
CESNET, z. s. p. o.
Zikova 4
Prague, Czech Republic
Email: pus@liberouter.org

Juraj Blaho, Jan Kořenek
Faculty of Information Technology
Brno University of Technology
Božetěchova 2, Brno, Czech Republic
Email: xblaho00@stud.fit.vutbr.cz, korenek@fit.vutbr.cz

***Abstract*—Packet classification algorithms are widely used in network security devices. As network speeds are increasing, the demand for hardware acceleration of packet classification in FPGAs or ASICs is growing. Nowadays hardware architectures can achieve multigigabit speeds only at the cost of large data structures, which can not fit into the on-chip memory.**

We propose novel method how to reduce data structure size for the family of decomposition architectures at the cost of additional pipelined processing with only small amount of logic resources. The reduction significantly decreases overhead given by the Cartesian product nature of classification rules. Therefore the data structure can be compressed to 10 % on average. As high compression ratio is achieved, fast on-chip memory can be used to store data structures and hardware architectures can process network traffic at significantly higher speed.

I. Introduction

With the rapid development of computer networks, traffic filtering has become one of the first steps in securing any network or computer. Basic traffic filtering device is the firewall, which performs per-packet decision based on the given set of rules. As network speeds are increasing, the demand for the speed of packet classification algorithms is also growing.

The classification algorithm contains a set of rules ordered by priority. Each rule defines a condition for all significant packet header fields. These fields are usually: Source IP Address, Destination IP Address, Source Port, Destination Port, Protocol. A condition may be exact match, prefix match (usually for IP addresses), range match (for ports), or a wildcard (matching any value). The goal of a packet classification algorithm is to find the matching rule with the highest priority. The output of the algorithm is then the number of the matched rule.

Software solutions for the packet classification problem are available, but their performance is not sufficient for wirespeed processing in the highest-speed networks. Existing FPGA and ASIC architectures can achieve multigigabit speeds only at the cost of large data structures and they must deal with great memory overheads imposed by fields Cartesian product.

We propose a novel method that significantly reduces memory requirements for classification algorithms by fast pipelined processing with only small amount of logic resources. As the processing is pipelined, only system latency is increased, but the throughput is not affected. The proposed method utilizes classification rules structure and can be used for any decomposition-based classification algorithm, where the processing is split between the longest prefix match (LPM) and rule matching operations.

The rest of the paper is organized as follows: in the next section we discuss the related work and explain causes of the excessive memory overhead. Section III introduces our new method of lowering memory overheads of these algorithms, together with the sketch of the proof of the algorithm correctness. Experimental results of our work are summed up in Section IV, and Section V concludes the paper.

II. Related work

As the packet classification problem is inherently hard from a theoretical standpoint [1], a large number of hardware and software solutions [1], [4] have been proposed.

From the wide choice of available algorithms, we discuss only those which are related to our work. All of them belong to the family of decomposition-based methods. In decomposition methods, packet classification is divided into several steps. First step is the LPM operation, which is performed independently in each dimension. From the given set of prefixes with various lengths, the LPM algorithm finds the one that best fits the given full-length value. Range conditions (such as port ranges) in the ruleset are converted to prefixes, so that the LPM may be performed in all dimensions. Figure 1 shows the basic scheme of all Cartesian product algorithms.

The LPM operation is performed in IP packet routing, so it is well studied topic. In fact, routing table lookup operation is a classification in one dimension only – the destination IP address. Basic algorithm for the LPM is a trie – the tree algorithm processing one input bit at each tree level and returning the last valid prefix visited. Trie is often modified to process more input bits in each step and to reduce memory requirements. Popular example of such algorithm is the Tree Bitmap [3], but there are also many other solutions.

Result of the LPM stage is a vector of prefixes, each prefix is represented by unique number. After the LPM, all fields of the resulting LPM vector must be combined together to get the resulting rule number. Basic Cartesian product algorithm [6] precomputes a product table, which contains resulting rule

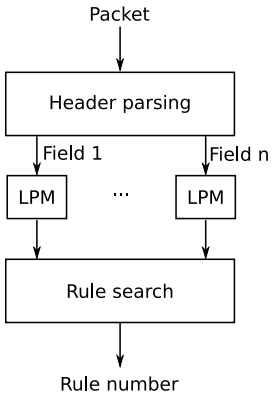

Fig. 1. Basic scheme of Cartesian product-based algorithms.

numbers for all possible combinations of prefixes. Because of the multiplicative nature of the Cartesian product, this table may become extremely large.

Multi Subset Crossproduct Algorithm [2] brings further improvements to decomposition methods. The authors of this work replace Cartesian products by *pseudorules*. Because pseudorules expansion is similar to Cartesian product, authors provide heuristics on how to break ruleset into several subsets, eliminating the majority of pseudorules. The paper also identifies rules that generate excessive amount of pseudorules. These rules are called *spoilers* and are moved to small on-chip TCAM.

We have recently published another Cartesian product-based algorithm [5]. Our method uses specifically constructed hash function to map all pseudorules (in the form of LPM result vectors) onto correct rules. This way, it is no longer necessary to store pseudorules, which saves a considerable amount of memory. The algorithm also achieves high packet rate, and the processing time for each packet is guaranteed to be constant. But still, the number of pseudorules affects the size of data structures of the perfect hash function. This means that even this memory-optimized algorithm may be significantly improved by reducing the number of pseudorules.

The algorithms mentioned in this section achieve very good speeds, but their memory requirements may be limiting and should be improved. Moreover, the general principle says that smaller memory technologies achieve higher speeds (small SRAM is faster than big DRAM), so that lowering the amount of required memory may bring further increase in speed.

Our goal is to design a new method applicable for all Cartesian product-based algorithms, as they all deal with the same memory explosion. The two memory optimization techniques introduced in [2] (spoilers removal and use of subsets) are direct competition to our method, but all memory reduction methods may be used together to produce even better results.

III. Memory optimization

We describe in detail how pseudorules are created and why this process increases the size of data structures in packet classification algorithms. To cover all valid combinations of LPM results, pseudorules must be added to the ruleset. In fact, a pseudorule is always a special case of some rule. This is best explained by the example of pseudorules generation in Fig. 2.

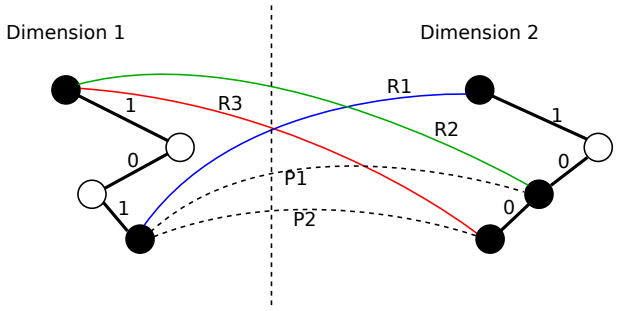

Fig. 2. Example rules and pseudorules.

Rule	Dimension 1	Dimension 2	Target rule
R1	101	*	
R2	*	10*	
R3	*	100	
P1	101	10*	R1
P2	101	100	R1

TABLE I
Example rules and pseudorules.

We can see classification in two three-bit dimensions with three rules. There is one trie for each dimension. Black circles represent valid prefixes (possible results of LPM operation in that dimension). There is, for example, no rule for packet with header fields $(101, 100)$, but the correct result is rule $R1(101, *)$[1]. Therefore, pseudorule $P1(101, 100)$ has to be added to cover this situation. Tab. I contains all rules and pseudorules together. *Target rule* in this table points to the correct classification result of the pseudorule. The generation of pseudorules has the character of Cartesian product, and it may expand the ruleset significantly

Our method is based on the observation that many classification rules often do not specify all the classification fields. For example, if user wants to block a specific source IP address, the rule does not specify destination IP address or port number. We use term *ANY* for these field conditions. This means that any destination IP and any port number can match this rule. However, the rule can create many pseudorules because all more specific destination IPs and ports have to be covered by pseudorules.

Fig. 3 is an example for two fields, where an ANY value in the rule produces many pseudorules. As the rule is quite *general*, we must deal with all more *specific* pseudorules. The situation is even worse for multiple fields.

[1]Symbol * denotes prefix or wildcard

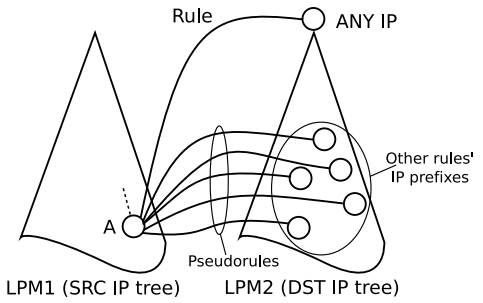

Fig. 3. One of the most severe causes of pseudorules: ANY values in the ruleset.

We address this issue and propose a solution to insert a *generalization stage* (GS) into the classification algorithm after LPM engines. The GS is able to replace LPM results with more general ANY value in certain situations. As a result, the number of output combinations is reduced after GS. This will result in smaller data structures of the following stages of all crossproduct algorithms.

The LPM result is compared to all values stored in the GS, and in the case of match, additional information instructs the GS to replace some of the other LPM results with the ANY value (to perform generalization). In the example from Fig. 3, if LPM1 returns value A, then the result from LPM2 is unimportant and may be replaced with more general ANY value. Thanks to this replacement, the number of possible GS outputs is reduced.

This concept is used generally for many fields, formally we can write: The *generalization rule (GR)* is a 3-tuple $R = (b, v, G)$ where b is an index to the vector of LPM results (which LPM result is to be compared), v is a value of particular LPM result and G is a set of indices to the vector of LPM results. The effect of one GR is: if $LPM[b] = v$, then foreach index $g \in G$ set $LPM[g] := ANY$. All GRs may be applied together, their ordering is unimportant.

This scheme corresponds to the following situation: we know that if a field $LPM[b]$ has a particular value v, then some other fields $LPM[g], g \in G$ are unimportant, because the result of classification is already determined.

It remains to find an algorithm to create GRs. First, we need a list of all pseudorules. As an input we have a list of classification rules ordered descending by priority. We need to traverse this list from the rule with the highest priority and for every rule create all corresponding pseudorules. For every classification rule, the most general pseudorule, which is that with the most ANY values, is created first. At the end we have a list of all pseudorules ordered by their priority.

This ordering helps us in the next part of the algorithm where GRs are created and some pseudorules are removed. The algorithm tries to identify situations when a classification rule defines value for a field with an index i and allows the ANY value in fields with indices G. Then in certain cases, for all LPM results with the same value at index i, values at indices G may be replaced by ANY value, and the result of the classification is still uniquely determined.

The algorithm traverses the list of pseudorules and tries to create new GRs. There are several conditions that must be met when creating a GR:

- The rule must contain at least one ANY value. This condition is not explicitly stated in the algorithm, because it is implicit: for rules with no ANY value, G would be empty and the GR would make no sense.
- The same value of field at index i has not appeared earlier in the list of pseudorules. If this condition is not true, we cannot ensure that the value in this field unambiguously determines the correct classification result.
- The ANY value has not appeared at the index i earlier in the list of pseudorules. The reason for this condition is the same as for the preceding one.

Each GR usually removes several pseudorules. If a GR removes no pseudorule, or only a few of them, it may be omitted without an effect on the classification correctness.

1: Input: List of pseudorules $list_p$.
2: Create empty set of generalization rules R.
3: **for all** pseudorules p_{this} from $list_p$ **do**
4: **for all** $i \in \{0..n\}$ **do**
5: **if** $p_{this}[i] \neq ANY$ and not exist previous $p_{prev} \in list_p$ such that $(p_{prev}[i] = p_{this}[i]$ or $p_{prev}[i] = ANY)$ **then**
6: Create new generalization rule $R_{new} = (i, p_{this}[i], G)$, where $G = \{j | p_{this}[j] = ANY\}$.
7: Add R_{new} to R.
8: Remove all pseudorules p_{after} that follow in $list_p$ after p_{this} where $p_{after}[i] = p_{this}[i]$ and exists $j \in G$ such that $p_{after}[j] \neq ANY$.
9: **end if**
10: **end for**
11: **end for**
12: Output: Set of generalization rules R, reduced list of pseudorules $list_p$.

A. Example

To demonstrate the function of our algorithm, consider rules and pseudorules from Tab. I and Fig. 2.

There are two pseudorules in this example, both of them are specific cases of the rule $R1$. But if LPM result in Dimension 1 is 101, then the result of classification is already unambiguously determined. Thus we create Generalization Rule $(1, 101, \{2\})$. The Generalization Stage with this GR will perform substitution of LPM results $(101, 10*)$ and $(101, 100)$ by $(101, *)$. Therefore, pseudorules $P1, P2$ are not necessary in the following steps of the classification algorithm and only the original rules need to be stored in this simple example.

B. Correctness

We start with the assumption that the classification algorithm without our optimization is correct. Namely we suppose that the *Rule search* step from Fig. 1 is able to obtain correct classification result from LPM results, with the knowledge of all pseudorules. Therefore we may suppose that the *Rule*

Ruleset	Rules	Pseu. before	Pseu. after	Ratio
real1	68	168 000	27 888	0.166
real2	335	44 153	2 197	0.049
real3	1194	114 826	19 600	0.170
real4	1529	2 584 281	63 546	0.024
synth1	47	42 500	11 570	0.272
synth2	472	4 985 457	263 085	0.052
synth3	962	3 205 517	949 205	0.296

TABLE II
RESULTS OF GENERALIZATION RULES SEARCH.

	Generalization rules in the field			
Ruleset	SRC IP	DST IP	SRC Port	DST Port
real1	18	0	0	7
real2	44	59	0	0
real3	12	0	0	0
real4	59	0	0	0
synth1	3	12	0	1
synth2	89	38	0	15
synth3	745	306	7	0

TABLE III
NUMBERS OF GENERALIZATION RULES IN SEPARATE FIELDS.

search step performs linear search in the list of rules and pseudorules, ordered by priority.

To prove the correctness of the generalization algorithm, we have to show that after processing LPM results in the GS, there is still enough information to obtain correct classification result, with the knowledge of the reduced set of pseudorules.

The condition in the step 5 of the algorithm means that if LPM result at the index i has the value $p_{this}[i]$, and the packet does not match any rule or pseudorule with higher priority, then the classification result is unambiguously known. This is because if the packet should be matched by a higher-priority rule, then it will be matched correctly during the (supposed) linear search in the *Rule search* step. Therefore, generalization is performed only if the classification result is unambiguously known.

All removed pseudorules must have the same or lower priority than the actual rule. The condition in the step 8 of the algorithm requires removed pseudorules to be specific cases of the actual rule. And because the correct result of the classification is already known, it is not necessary to store more specific pseudorules of this rule to classify the packet correctly.

IV. RESULTS

We performed analysis and Generalization Rules search for several real-life firewall sets from the university campus network, as well as synthetic ruleset generated by freely available ClassBench tool [7].

The GR search process is based on multiple ANY values in classification rules. The compression ratios for all mentioned rulesets are shown in Table II. Number of pseudorules before and after reduction are compared. Table III shows numbers of GRs in each dimension. It can be seen that our method reduces the number of pseudorules significantly, even if only several GRs are added. Moreover, some fields don't have any generalization rule. This observation may be used to reduce the number of FGEs and simplify hardware implementation of the whole GS. For example only three FGEs are needed to implement GS for all rulesets in Table III.

V. CONCLUSION

In this paper, we propose a novel method how to significantly reduce memory resources for fast packet classification. The presented method can be used for any decomposition-based classification algorithm where the processing is split between the longest prefix match and a rule matching operations. In addition, the proposed method is orthogonal to other existing memory reduction approaches and provides further reduction in memory needs.

The proposed reduction method achieves compression ratio up to 0.024 and 0.1 on average for rulesets available to us. It means that the memory size can be reduced 10 times on average and about 50 times in the best case. The results depend on the size of the ruleset and its structure. After the reduction, small rulesets can fit into the faster on-chip memory and the classification algorithm can be speed-up.

Our future work will focus on improving a compression ratio. As can be seen in the Table II, Synth1 and Synth3 rulesets exhibit lower compression despite the large amount of ANY fields. Our aim is to explore reasons leading to lower compression and address them to improve proposed method.

ACKNOWLEDGMENT

This research has been partially supported by the Research Plan No. MSM, 6383917201 – Optical National Research Network and its New Applications, the Research Plan No. MSM, 0021630528 – Security-Oriented Research in Information Technology, and the grant BUT FIT-S-10-1.

REFERENCES

[1] F. Baboescu, S. Singh, and G. Varghese. Packet classification for core routers: Is there an alternative to CAMs? In *INFOCOM*, 2003.

[2] S. Dharmapurikar, H. Song, J. Turner, and J. Lockwood. Fast packet classification using Bloom filters. In *ANCS '06: Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, pages 61–70, New York, NY, USA, 2006. ACM.

[3] W. Eatherton, G. Varghese, and Z. Dittia. Tree bitmap: hardware/software IP lookups with incremental updates. *SIGCOMM Computer Communication Review*, 34(2):97–122, 2004.

[4] P. Gupta and N. McKeown. Algorithms for packet classification, 2001.

[5] V. Puš and J. Kořenek. Fast and scalable packet classification using perfect hash functions. In *FPGA '09: Proceedings of the 17th international ACM/SIGDA symposium on Field programmable gate arrays*, New York, NY, USA, 2009. ACM.

[6] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. *SIGCOMM Comput. Commun. Rev.*, 28(4):191–202, 1998.

[7] D. E. Taylor and J. S. Turner. Classbench: a packet classification benchmark. *IEEE/ACM Trans. Netw.*, 15(3):499–511, 2007.