BitCuts: Towards Fast Packet Classification for Order-Independent Rules

Zhi Liu[1, 2], Xiang Wang[1, 2], Baohua Yang[3] and Jun Li[2, 4]
[1]Department of Automation, Tsinghua University, China
[2]Research Institute of Information Technology, Tsinghua University, China
[3]IBM China Research Lab, Beijing, China
[4]Tsinghua National Lab for Information Science and Technology, China
{zhi-liu12, xiang-wang11}@mails.tsinghua.edu.cn, baohyang@cn.ibm.com, junl@tsinghua.edu.cn

CCS Concepts
• **Networks~Packet classification**

Keywords
Packet Classification; Order-Independent Rules

1. Introduction
Packet classification is required to achieve high throughput while fitting into a commodity memory hierarchy. Therefore, fewer memory accesses and a reasonable memory footprint are the main concerns when designing a packet classification algorithm. Recent work proposed a hybrid packet classification solution incorporating both software and TCAM-based approaches. Specifically, SAX-PAC[1] observed that real-life packet classification rules can be represented by a few groups of *order-independent* rules. In each group, any two rules do not intersect and can possibly be separated on a fraction of fields, reducing the complexity of the classifier. Consequently, it has been suggested that the *order-independent* groups be handled by existing software algorithms, and the rest of the rules (the *order-dependent* part) be handled by TCAM. However, SAX-PAC did not design specific software algorithms for *order-independent* rules. For these rules, existing algorithms are still inefficient in terms of classification speed. Decision-tree algorithms, the state-of-the-art software approach, always traverse tens of tree nodes in order to identify the matching rule. Speed is further decreased when packets need to match multiple *order-independent* groups.

This paper proposes BitCuts, a bit-level decision-tree algorithm targeting a promising classification speed on *order-independent* rules. Our evaluation shows that BitCuts only requires 30%~40% of the number of memory accesses of existing algorithms and still achieves small a memory footprint on large rulesets.

2. Bit-level Separability of Order-Independent Rules
Consider an example of three order-independent rules in Figure 1, each specifying arbitrary ranges on three fields. These rules are "order-independent" since each pair of the rules does not intersect. Also, the first two fields (bits 0-3 and 4-7, as shown in the dashed boxes) can guarantee this property. Figure 1 also illustrates how bits classify order-independent rules efficiently. Field ranges are converted to prefixes for ease of understanding.

SIGCOMM '15, August 17-21, 2015, London, United Kingdom
ACM 978-1-4503-3542-3/15/08.
http://dx.doi.org/10.1145/2785956.2789998

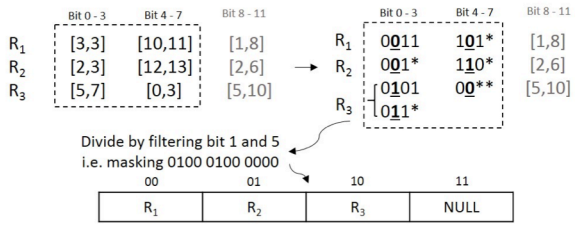

Figure 1. Example ruleset classified by 2 bits

It is shown that a classifier based on bits 1 and 5 is sufficient to separate all of the rules. These bits partition the rules into four subsets. The resulting subsets are stored in an array of buckets, where each bucket is indexed by the concatenated bit values and contains a pointer to the actual rule. It is found that the rules are "fully separated" in these buckets, meaning that each bucket contains at most one rule.

For example, to classify a packet with header {0011, 1101, 1111}, the classification process includes: ***Masking*** the packet header fields with {0100, 0100, 0000} and filtering out the bits at non-zero mask positions, which is "01" in this case; ***Indexing*** to bucket "01" and conducting a "false positive check" with all fields to verify if it matches with rule R_2. In the worst-case, the classification only requires one access to the root bucket, one to the leaf, and a final access to the possible matching rule, dramatically reducing the number of memory accesses.

It can be proved that all of the rules in an order-independent group can be "fully separated" by selecting an appropriate set of bits [2]. To adopt this property for larger rulesets, a naïve solution is to find l bits that "fully separates" the rules and construct the corresponding 2^l buckets, as shown in Figure 1. Table 1 shows the number of required bits l to "fully separate" our evaluated order-independent rulesets, generated with the heuristic proposed in [1]. However, the number of required bits is always prohibitively high, making this solution impractical due to the memory footprint of the 2^l buckets.

3. BitCuts Design
To reduce the number of memory accesses as well as achieve a moderate memory size, we propose the "BitCuts" approach. BitCuts fully utilizes the bit-level separability of order-independent rules and constructs a multi-layer decision-tree to classify incoming packets. An example of resulting data structure and classifying procedure is shown in Figure 2. BitCuts first accesses the root bucket, fetches a bitmask with 10 selected bit positions, and masks the packet header against this bitmask. The concatenated bit values index to another intermediate bucket on the next layer, which contains a bitmask and a bucket array pointer for the next access. This recursion stops until it reaches a leaf bucket, and then a false positive check is conducted to get

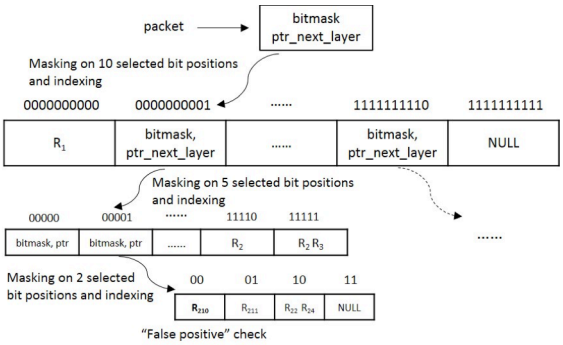

Figure 2. Data structure and packet classification procedure of BitCuts

Algorithm 1 – BitCuts_Tree_Build

```
1   function BitCuts_Tree_Build(ruleset)
2     bitmask = all_zeros()
3     do:
4       bit = bit_select(ruleset, bitmask)
5       bitmask = bitmask.add(bit)
6       subsets = partition(ruleset, bitmask)
7     While(insep_fraction_calculate(subsets) > threshold)
8     L = bitmask.number_selected_bits
9     bucket_array = initialize(pow(2, L))
10    for each i in subsets.indexes:
11      if inseparable(subsets[i]):
12        bucket_array[i].is_leaf = True
13        bucket_array[i].ptr = get_ptr(subsets[i])
14      else:
15        p_next_layer_bucket_array, bitmask =
                BitCuts_tree_build(subsets[i])
16        bucket_array[i].is_leaf = False
17        bucket_array[i].bitmask = bitmask
18        bucket_array[i].ptr = p_next_layer_bucket_array
19    return addr(bucket_array), bitmask
```

the final match. In a nutshell, BitCuts' multi-layer design has the following advantages over the naïve solution:

Better rule separation. On upper layers, each subset in the bucket array always requires a different set of bits in order to be divided further. With the constraint of maintaining a reasonable memory size, BitCuts' multi-layer design is able to select the best bits to divide each subset, thus making use of more bits than naïve solution.

Smaller memory size. The size of each subset varies among different buckets. Some of the subsets are relatively small and might not need additional bits for further dividing, while some might be large and require more bits. Our multi-layer design will prevent the space inflation resulting from unnecessarily dividing very small subsets.

The pseudocode for building BitCuts decision-trees is shown in Algorithm 1. The algorithm iteratively calls *bit_select* and adds the selected bit to the bitmask. Currently, *bit_select* uses a greedy strategy: it tries to add one unused bit to the existing bitmask, splits the ruleset, and gets the maximum size among all the resulting subsets. After a trail with each unused bit, the greedy algorithm selects the bit with the lowest maximum size.

As shown in line 8, the bit selection stops when the fraction of "inseparable" subsets, which will be explained later, exceeds the pre-defined threshold. If L bits are selected at this point, the algorithm checks each of the resulting 2^L subsets. If the subset is "inseparable", the algorithm constructs the corresponding leaf bucket. If not, the subset needs to be divided further by constructing a bitmask and bucket array for the next layer.

The goal of calculating the fraction of "inseparable" subsets (*insep_fraction_calculate*) is to determine when to stop the bit selection on the current layer, which is fundamental to achieve a good trade-off between classification speed and memory size. To figure out whether a subset is "inseparable", an obvious criterion is that buckets with no more than 1 rule are naturally "inseparable". On the other hand, adding one additional bit to further divide small subsets is not efficient since it doubles the bucket number on the current layer but provides limited savings in the number of memory accesses. Currently we judge a subset as "inseparable" if its size is no more than 2.

Ruleset	*Coverage*	*# of bits*	*BitCuts*			*HyperCuts*			*HyperSplit*		
			Max	Avg	Size (KB)	Max	Avg	Size (KB)	Max	Avg	Size (KB)
acl5k_0	72.45%	70	6	4.2	102	19	9.1	51	16	12.1	65
acl5k_1	18.08%	49	7	3.9	155	35	10.4	20	13	10.2	16
acl10k_0	78.69%	55	7	4.0	278	19	11.0	199	16	13.2	149
acl10k_1	10.49%	30	6	3.9	16	17	9.6	10	12	10.2	19
fw5k_0	68.30%	53	6	4.1	41	18	10.4	88	15	11.9	62
fw5k_1	24.54%	23	5	4.2	16	17	10.4	42	12	10.4	22
fw10k_0	74.61%	12	4	3.7	43	18	11.3	252	14	13.0	135
fw10k_1	24.27%	11	5	4.1	33	17	11.3	97	13	11.3	44
ipc5k_0	52.74%	79	7	4.1	191	19	9.7	184	16	11.7	48
ipc5k_1	24.21%	62	6	4.1	58	21	9.7	29	13	10.6	23
ipc10k_0	53.46%	86	7	3.8	300	21	10.4	307	16	12.7	99
ipc10k_1	26.24%	66	6	4.2	103	24	10.3	71	15	11.7	51

Table 1. Memory access count (worst-case and average-case) and memory size comparison on order-independent groups of ClassBench rulesets

4. Evaluation

We evaluate BitCuts on order-independent groups generated by the algorithm proposed in [1] and compare the performance with two other decision-tree algorithms, HyperCuts [3] and HyperSplit [4]. Table 1 shows the memory access count and the size of the resulting data structure. It's shown that BitCuts greatly reduce the number of memory accesses. In the worst case, the BitCuts memory access count is ~42% that of HyperSplit and ~30% that of HyperCuts. On average, the BitCuts memory access count is only ~35% that of HyperSplit and ~40% that of HyperCuts. When comparing memory size, BitCuts is 2.3x that of HyperSplit and 1.6x that of HyperCuts, on average. For the evaluated 10k rules, the memory size is still reasonable and the data structures could easily fit into the cache. For rulesets that require fewer bits (e.g. order-independent groups 0 and 1 of fw10k), BitCuts achieves the lowest memory size because it requires much fewer intermediate nodes.

5. Conclusion and Future Work

To accelerate software-based classification for order-independent rules, this work proposes BitCuts, a bit-level decision-tree algorithm for fast packet classification. BitCuts is proved promising to achieve much fewer memory accesses compared to existing algorithms while maintaining a reasonable memory footprint. Our future work includes efforts to improve the efficiency of the bit selection algorithm, and further evaluations on real traffic.

6. References

[1] K. Kogan, S. Nikolenko, O. Rottenstreich, W. Culhane, and P. Eugster. SAX-PAC (Scalable And eXpressive PAcket Classification). In ACM SIGCOMM 2014.

[2] B. Yang, J. Fong, W. Jiang, Y. Xue and J. Li. Practical Multituple Packet Classification Using Dynamic Discrete Bit Selection. In IEEE TRANSACTIONS ON COMPUT-ERS, VOL. 63, NO. 2.

[3] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet Classification using Multidimensional Cutting. In ACM SIGCOMM 2003.

[4] Y. Qi, L. Xu, B. Yang, Y. Xue and J. Li. Packet Classification Algorithms: From Theory to Practice. In IEEE INFOCOM, 2009.