Structural Compression of Packet Classification Trees

Xiang Wang[1, 2], Zhi Liu[1, 2], Yaxuan Qi[2] and Jun Li[2]
[1]Department of Automation, Tsinghua University, China
[2]Research Institute of Information Technology, Tsinghua University, China
{xiang-wang11, zhiliu08}@mails.tsinghua.edu.cn, {yaxuan, junl}@tsinghua.edu.cn

ABSTRACT

Most of state-of-the-art packet classification algorithms employ heuristics to trade off between classification speed and memory usage. However, intelligent heuristics often result in complex data structures in algorithm implementation. This brings difficulties to the deployment and optimization of packet classification algorithms. In this poster, a structural compression approach is presented for decision tree based packet classification algorithms. This approach exploits the similarity in real-life filter sets to achieve high compression ratio without loss of tree semantics.

1. INTRODUCTION

Packet classification has been studied for a long period, and many decision tree based packet classification algorithms have been proposed. Most of them trade worst-case search time for memory space. They implement various strategies of search space decomposition, because consistent cuttings on the other hand usually introduce excessive overhead of memory usage. For example, several algorithms set *binth* [1, 2] to control the load of linear search at leaf nodes, and take variable-stride cuttings at internal nodes. Those techniques not only hamper the guarantee of worst-case classification time among all types of filter sets, but also limit the optimization of search data structure for hardware acceleration.

In this poster, we argue that the consistent cutting strategy is the key to the improvement of both processing speed and memory usage. The memory overhead of consistent strategy is categorized into two types of redundancy: the *local redundancy* usually exists in pointer arrays in each internal node, and the *global redundancy* lurks in the space decomposition model among all internal nodes.

This poster investigates those redundancies in fixed-stride cutting trees, and illustrates the method to remove them individually.

2. ALGORITHM

Structural redundancy can be illustrated based on a typical HiCuts [1] tree. Figure 1 shows a 4-rule classifier and its corresponding HiCuts Tree. Most of implementations of HiCuts and similar algorithms use pointer array to address child nodes and store the mapping between unit-spaces of fixed-stride cutting and aggregated sub-spaces. It is obvious that direct pointer addressing scheme brings too much overhead of memory usage, as each pointer takes one word length memory, which is regarded as local redundancy.

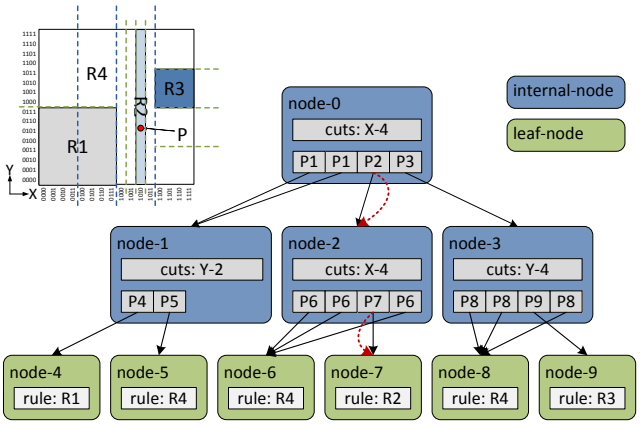

Figure 1. 4-rule Classifier and HiCuts Tree

Furthermore, due to the similarity of the rule distribution in the whole search space, several nodes share the same space decomposition performed in their own sub-spaces. For example, node-2 and node-3 have the same numbers of both unit-space and sub-space. Besides, they both aggregate the 1st, 2nd and 4th unit-space to the 1st sub-space, and map the 3rd unit-space to the 2nd sub-space. Each internal node needs to store the mappings, which is regarded as global redundancy. We propose a 3-step structural compression algorithm to eliminate the redundancy. The first step removes the pointer array. The second step compresses the local redundancy by using a bitmap technique. And the third step extracts the space decomposition from all nodes, which is further aggregated into two shared memory.

STEP-1: Elimination of pointer array

We argue that it is the pointer array that hampers the reduction of both local and global redundancy. To reduce the overhead introduced by the pointer array, the tree building procedure bounds each tree node into fixed size and stores nodes in consecutive memory. All nodes sharing the same parent are viewed as siblings. The parent node only needs to store its first child node address, and uses offset to address other child nodes. As a consequence, the original pointer array can be expressed using one base pointer and one offset array. If the procedure takes 256-stride cutting strategy, each array element can be stored within single byte. The significance of this step is not only reducing the overhead of storing pointers, but also providing the possibility of compressing the global redundancy.

STEP-2: Elimination of local redundancy

In the previous step, each node uses a base pointer with an offset array to address its child nodes, and the offset array takes the majority of memory space in each node. In practice, vast internal nodes only have fewer child nodes, which mean many consecutive unit-spaces will be aggregated into the same sub-space [3]. As a consequence, the bitmap technique is employed to compress the offset array, which generates a bitmap and a compressed offset list. Figure 2 shows the classification tree which is ready for the third compression step.

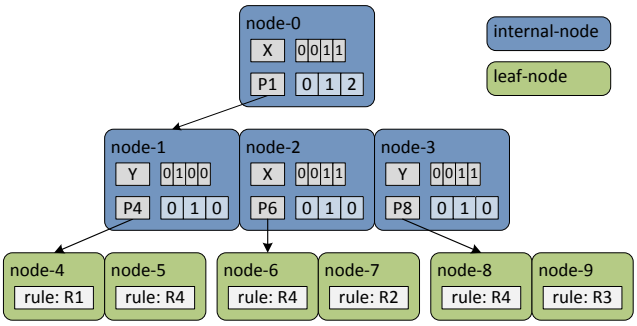

Figure 2. Classification Tree Ready for Elimination of Global Redundancy

STEP-3: Elimination of global redundancy

After the formal two steps, it is observed that a great deal of internal nodes has same bitmaps and offset lists, and the number of unique bitmap and offset list are both relative small in practical classifier. Based on this observation, unique bitmaps and offset lists are extracted and stored in two consecutive shared memories respectively, leaving two indices in each internal node.

After the 3-step structural compression, the original classification tree is reshaped into three lookup tables, and a compact packet classification tree is obtained. Figure 3 shows the compressed search data structure of classifier in Figure 1.

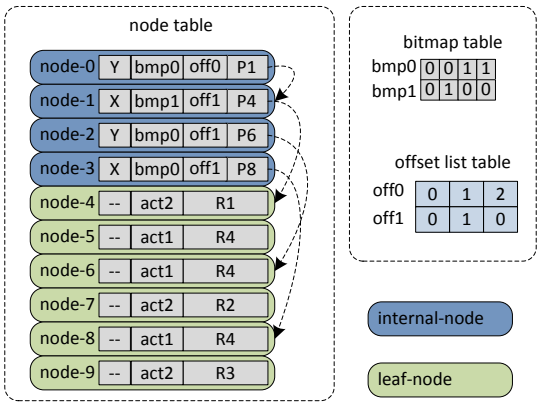

Figure 3. Classification Tree for Structural Compression

3. EVALUATION

We carry out the preliminary evaluation test on FW classifier generated by classbench [4]. Table 1 compares the numbers of bitmap and offset before and after elimination of global redundancy. Figure 4 shows the memory size comparison between original HiCuts tree, AggreCuts [3], HyperSplit [2] and structural compressed tree. When bounding to the same memory access time, HiCuts tree needs about 2 orders memory usage than structural compressed tree. Besides, the memory of structural compressed tree is comparable to the one of HyperSplit, where the memory access time of the latter is two times larger than the former one.

Table 1. Global Redundancy of Bitmap and Offset List

rule	*FW_100*	*FW_1K*	*FW_5K*	*FW_10K*
total	6012	304185	1585.3K	4818.4K
uni bmp	116	703	2612	3713
uni off	23	109	293	499

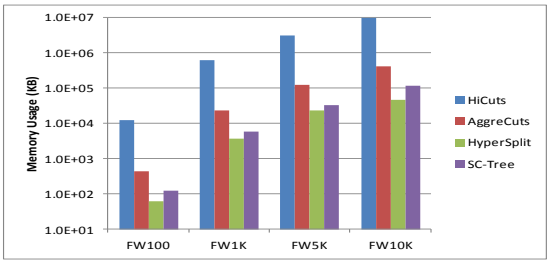

Figure 4. Memory size of HiCuts, AggreCuts, HyperSplit and structural compressed tree

4. CONCLUSION

This poster presents a structural compression approach to eliminate redundancy in packet classification trees. It exploits the similarity residing in real-life filter sets, and achieves high compression ratio in preliminary evaluation. Our future work will apply structural compression to other equal-sized space decomposition packet classification algorithms with grouping of rules [5, 6].

5. REFERENCES

[1] P. Gupta and N. McKeown. "Classifying Packets with Hierarchical Intelligent Cuttings," in IEEE Micro, 2000.

[2] Y. Qi, L. Xu, B. Yang, Y. Xue and J. Li. "Packet Classification Algorithms: From Theory to Practice," in Proc. of INFOCOM, 2009.

[3] Y. Qi, B. Xu, F. He, B. Yang, J. Yu and J. Li. "Towards High-performance Flow-level Packet Processing on Multi-core Network Processors," in Proc. of ANCS, 2007.

[4] ClassBench: A Packet Classification Benchmark. http://www.arl.wustl.edu/classbench/

[5] B. Vamanan, G. Voskuilen and T. Vijaykumar. "EffiCuts: Optimizing Packet Classification for Memory and Throughput," in Proc. of SIGCOMM, 2010.

[6] J. Fong, Y. Qi, J. Li and W. Jiang. "ParaSplit: A Scalable Architecture on FPGA for Terabit Packet Classification," in Proc. of HOTI, 2012.