Proceedings of the IASTED International Conference
Communication, Network, and Information Security
October 9–11, 2006, MIT, Cambridge, MA, USA
ISBN CD: 0-88986-636-8

AN EFFICIENT HYBRID ALGORITHM FOR MULTIDIMENSIONAL PACKET CLASSIFICATION

Yaxuan Qi[1] and Jun Li[1,2]
[1] Research Institute of Information Technology (RIIT), Tsinghua University, Beijing, China, 100084
[2] Tsinghua National Lab for Information Science and Technology (TNLIST), Beijing, China, 100084
{yaxuan, junl}@tsinghua.edu.cn

ABSTRACT
Multidimensional Packet Classification is one of the most critical functions for network security devices such as firewalls and intrusion detection systems. Due to the worst case bounds found in computational geometry, most of the existing algorithms for multidimensional packet classification trade memory usage for search speed in order to achieve better overall performance. Although some of these algorithms are proved to be efficient on small number of classification rules, they scale poorly in either search time or memory usage when the number of rules grows. In this paper, we propose an efficient hybrid algorithm named sBits, which combines the advantages of two best existing algorithms, RFC and HiCuts. Compared to RFC and HiCuts, sBits uses 10 to 400 times less memory storage and 30% to 50% less time in worst case search. sBits also reduces the heavy computational burden in pre-processing. Its full update time is 10 to 100 times less than RFC and HiCuts.

KEY WORDS
Network security, Packet classification, ACL and IDS

1. Introduction

Keeping network operation and information exchange secure and efficient is highly desired in today's Internet communication. A variety of security services such as access control in firewalls and protocol analysis in IDS require a discrimination of packets based on the multiple fields of packet headers, which is called *multidimensional packet classification.*

Although there has been quite a few papers published on multidimensional packet classification [1, 3, 4, 5, 6, 7, 8, 9, 11, 13, 20] in recent years, researchers in both academic and industry continue to seek better solutions due to the ever increasing importance and requirements in today's high performance policy enforcing networks. The need for novel algorithms comes:

a. Hardware Limits: Currently, to obtain multi-Gbps multidimensional packet classification rate, there are only a few ASIC/FPGA products. Although hardware-based devices offer a good solution for application with small number of rules, such as those using TCAM, they consume too much electric power and board area for large rule sets [8]. In addition, hardware solutions usually mean higher cost for R&D and production, and lower flexibility in term of modification or upgrade. Therefore, it is worth looking for alternative solutions to overcome the restrains in hardware solutions.

b. Performance Limits: Due to the worst case bounds in computational geometry, most multidimensional packet classification algorithms trade memory usage for search speed in order to achieve better overall performance. However, even the best reported algorithms [1, 7] fail to provide ideal performance when tested on some real-life rule sets with large number of rules [19].

In this paper, we propose an efficient packet classification algorithm that outperforms the best published results in recent literatures [1, 3, 4, 7]. Main contribution of this paper includes:

a. New Methodology for Algorithm Analysis: Different from the previous *descriptional* and *dissectional* algorithm comparison and categorization in [16, 17, 19], we proposed a refined generic framework to analyze existing algorithms. Such an analysis helps to find more efficient hybrid algorithms that leverage on the advantages of other popular algorithms to reach much higher level of performance.

b. An Efficient Hybrid Algorithm: In this paper, we introduce an efficient hybrid algorithm named *Shifting Bits, sBits* in short. The overall data structure constructed by *sBits* is a decision tree similar to those of HiCuts [1] and its variations [7, 14]. While at each internal node an indexing mechanism is adopted, which can be viewed as an extension of the lookup tables used in RFC [3] and its variation [4].

c. Comprehensive Experiments and Evaluations: Software implementation of *sBits* and other popular algorithms [1, 3, 4, 7] has been developed with our best effort to make sure the fairness of our result comparison and analysis in experimental study. Thorough comparisons are done with several real-life rule sets, as well as synthetic ones. Experimental results include worst case search time, overall memory usage, full update time, and scalability on large rule sets.

The rest of the paper is organized as follows. Section 2 states the problem of packet classification; Section 3 analyzes prior work; Section 4 describes the proposed algorithm *sBits*; Section 5 illustrates the experimental results; as a summary, Section 6 states our conclusions.

2. Problem Definition and Complexity

Multidimensional packet classification classifies a packet based on multiple fields of the packet header.

Mathematically, a packet P is said to *match* a particular rule R, if the i^{th} field of the header of P satisfies the regular expression $R[i]$, for all $0 \le i < F$. If a packet P matches multiple rules, the matching rule with the highest priority is returned.

Multidimensional packet classification can be viewed as a point location problem in computational geometry, which is inherently hard to solve [16]. It has been proved that the best bounds for point location in N non-overlapping F-dimensional hyper-rectangles are $O(N)$ storage space with $O(\log^{F-1} N)$ search time, or $O(\log N)$ search time with $O(N^F)$ storage space [2]. However, in multidimensional packet classification problems, rules (hyper-rectangles in the multidimensional search space) may overlap, making classification at least as hard as point location. Moreover, the large constant hidden in the $O(\cdot)$ notation also impacts actual performance severely in practical implementation [19].

Although the theoretical bounds make it impossible to design a single algorithm that performs well for *all* cases, fortunately, real-life rule sets have some inherent characteristics that can be exploited to reduce the complexity in both search time and storage space [19]. There have been various statistics and characteristics of real-life rule sets presented and also exploited in the proposed algorithms [3, 8, 9, 17, 18]. Some algorithms like RFC and HiCuts achieve promising results in comparison to prior schemes. Other research, such as [11, 14], make efforts to take more factors (e.g. word width, adjustable constants) into consideration so as to give more precise theoretical bounds. Research has also been carried out in introducing traffic flow statistics in addition to rule statistics for using more heuristics that help in classification [13, 15]. All these studies provide us with thorough understanding of the existing multi-dimensional packet classification algorithms and hence motivate our research in this paper.

3. Analysis of Prior Work

3.1 The Framework for Algorithm Analysis

Surveys and taxonomies of prior work on packet classification [16, 17] break the design space of existing algorithms based on the high-level approach, such as exhaustive search, geometric tries and heuristic algorithms. This kind of categorization is helpful for algorithm description, because algorithms fall in the same category have similar data structures. Different from such a *descriptional* taxonomy, a *dissectional* one is also suggested that categorised packet classification algorithms according to their differences in space decomposition schemes and classifier data structures [19]. Such a *dissectional* taxonomy unveils the cohering relation lying in different algorithms. To balance the advantages of both methodologies, in this paper, we adopt a *Divide-and-Conquer* strategy in the study of existing algorithms. *Divide* means the partition of the search space and its corresponding rule set, while *Conquer* refers to the packet searching strategies. Two generic procedures can be deduced from most existing algorithms:

a. Partition the Search Space: In this procedure, the search space is partitioned into certain number of sub-spaces. Each sub-space is allocated with a subset of rules to create a new but scaled down search problem. By recursively partition the search spaces, as well as the corresponding rule sets, the complexity of the original classification problem is reduced, so the search result can be obtained by solving a series of sub-problems instead of doing exhaustive search in the entire search space with all rules.

b. Search the Packet: This procedure traverses the data-structure constructed by a particular algorithm to obtain the classification result. More specifically, packet search answers the questions of how to locate a point into its corresponding sub-space at each stage and how to go from current search stage to the next. Data structures of different classifiers, such as decision trees and lookup tables, represent different techniques adopted by each algorithm in the search procedure.

The following part of this section analyzes how the different implementation of these two procedures in existing packet classification algorithms.

3.2 Space Partition

There are two types of partition techniques adopted by existing algorithms: One uses geometric tries or trees (trie-based) while the other bases on rule projections (projection-based).

Trie-based algorithms partition the search space into 2^w equal-sized sub-spaces at each stage, where w refers to the *stride*, determining the number of sub-spaces. More specifically, binary tries, adopted in [5, 6, 8], partition the search space into 2 sub-spaces (w=1) at each stage, while multi-bit tries partition the search space into 2^w (w>1) sub-spaces. Decision trees in [1, 7, 11, 14] can be viewed as multi-bit tries with various stride in different level (w variable). Binary trie has at least $O(\log_2 |U|)$ in query time, where U denotes the entire search space (the *Universe*) and $|U|$ is the full range covered by U, e.g. for the common 5-field search, binary trie needs about 100 memory accesses. Multi-bit tries improve the search time to $O(\log_2 |U| / w)$ but tends to require larger storage.

Because trie-based algorithms implement uniform partitions, a single rule (specified by ranges) might be cut into fragments in multiple dimensions. There are two ways to allocate rules for sub-spaces: Range-to-prefix mapping for longest prefix matching and rule-duplication for range matching. For binary tries like Hierarchical Trie [16] and Grid-of-Trie [5], the sub-space at each node can be described by a prefix; hence the sub-space itself represents the longest matching prefix (rule). For multi-bit tries like decision trees in HiCuts and HyperCuts, each sub-space is allocated with a *colliding rule set*, i.e. all rules collide with the sub-space. To reduce the rule redundancy, G-filters allocate each sub-space with

different type of ruleset: *cross set*, *fallback set* and *cover set* [14].

Projection-based algorithms partition the search space according to the rule projection on each dimension. This technique is adopted by the algorithms proposed in [3, 4]. Projection-based methods have less redundancy in space partition because each segment is well cut to fit the rules. There is no need for further partition of each segment.

It is relatively simple to allocate rules for algorithms using projection-based partition, because each segment (representing a sub-space) is *fully covered* by the projection of particular rules. Hence this set of rules is allocated to the sub-space.

3.3 Packet Search

Most algorithms partition the search space along a single dimension at each stage. Algorithms that simultaneously partition the search space in multiple dimensions [7, 14] can be viewed as a multi-step single-dimensional partition (because all the separating hyper-planes are parallel to the coordinate axes). Therefore, each algorithm should answer how to locate a point into its corresponding sub-space along each dimension and how to link the sequential search stages going through multiple dimensions.

Trie-based algorithms works well for packet search due to equal-sized partition. At each node, the 2^w sub-spaces are associated with a pointer index of 2^w entries. These pointers connect the sequential nodes and lead the way for search. To locate the packet into its corresponding sub-space, trie-based algorithms need $O(1)$ time at each node.

It is relatively more complicated to locate a packet in projection-based partition. Because each segment may have different size, the search space is partitioned non-uniformly. It requires $O(\log_2 N)$ query time to do a binary search along each dimension to locate the packet in the corresponding segment [4]. P. Gupta in [3] suggested an indexing table to store the rule segments IDs and achieved $O(1)$ query time, but this scheme is hard to implement when index tables contain a very large number of entries, such as the IP address in IPv6.

As a summary, Figure 1 shows the generic framework for categorization of existing packet classification algorithms.

4. The Proposed Algorithm

We follow the generic procedures in designing the new algorithm. In this paper, we propose a novel algorithm which partitions the search space using a fixed stride decision tree and performs packet search with extended ID indexes. Due to the fixed-stride, the algorithm checks each w bits in the packet header at each node, so we call the proposed algorithm Shifting Bits, or sBits in short.

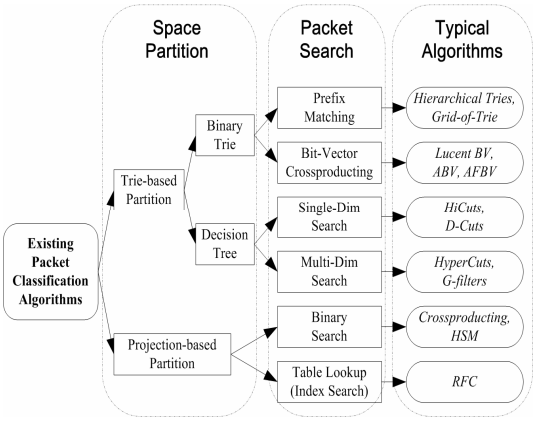

Figure 1. Categorization of existing multi-dimensional packet classification algorithms.

4.1 Space Partition with Decision Tree

At each internal tree node, space partition can be applied on both single and multiple dimensions. Multi-dimensional space partition is proved to be more effective both in worst case search speed and memory usage. *sBits* partitions the search space on the most discriminative dimensions, and the dimension selection mechanism is similar to HiCuts and HyperCuts [1, 7].

Trie-based partition uniformly divides the search space into 2^w sub-spaces. A larger stride w, i.e. larger number of sub-spaces, can cut down the depth of the tree while need more increase the storage requirements. To ensure the worst case search time, we use a big stride w. In the implementation of *sBits*, we set w=4~8.

In this section, we describe the space partition scheme by a 2-D sample rule set, which is shown in Figure 2.

First, the search space is partitioned into 2^w equal-sized sub-spaces on each dimension. Each intersection of these sub-spaces is called a *unit-space* because it is the minimum unit can be discriminated in current stage. Figure 3 describes the equal-sized partition and the consequent unit-spaces respectively.

Then, adjacent unit-spaces are aggregated if they contain the same colliding rule set along each partitioned dimension. Figure 4 and Figure 5 illustrate the space aggregation result for the 2-D example.

Actually, after the aggregation step, the current search space is partitioned into sub-spaces with colliding rules and hence initiates new search problems.

4.2 Packet Search with ID Index

sBits recursively partitions the search space into sub-spaces and constructs a decision tree to link each search space with its sub-spaces. The search of a packet P is to trace down the tree from root node to one of the leaf nodes. Thus, the construction of an appropriate data structure for fast packet location in its corresponding sub-space becomes the next key issue in packet search.

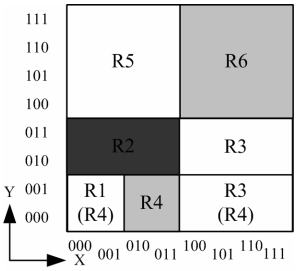

Figure 2. A Sample Rule Set with six 2-D rules in the X-Y plane. Each rule appears to be a rectangle in the search space. R4 is overlapped by R1 and R3.

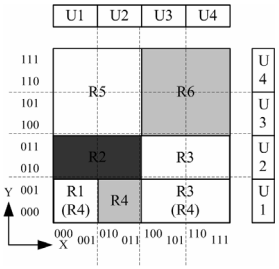

Figure 3. Unit-Spaces. The search space (X-Y plane) is partitioned into 4x4=16 unit-spaces; each has the same size of 2x2 squares in the X-Y plane.

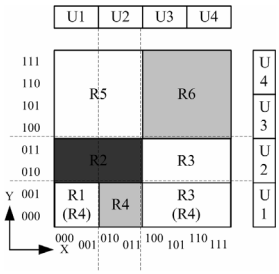

Figure 4. Aggregation of Unit-spaces. Along both X and Y dimensions, unit-space with the same rule projection are aggregated.

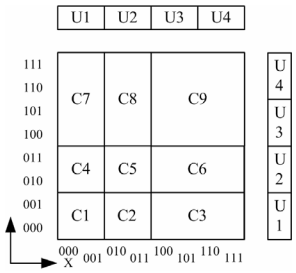

Figure 5. Child Nodes. For example, child node C3 corresponds to sub-space of {[100,111], [000,001]} associated with R3 and R4.

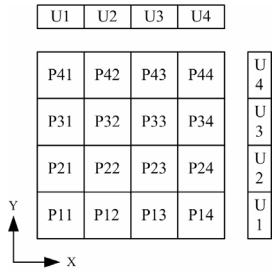

Figure 6. Pointer Matrix. The 4x4 pointers map each of the 16 unit-spaces into one of the 9 child nodes (C1~C9). For example, if a packet drops in the unit-space [U2, U4], then P24 will lead the way to C6 for further search.

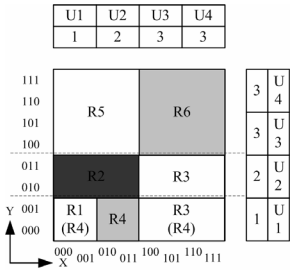

Figure 7. ID Indexes. The 2x4 space IDs map the 16 unit-spaces into 9 child nodes. If one packet drops in the unit-space [U2, U4], then by computing (2-1)*3+3=6, we know that this packet belongs to child node C6.

A direct way to link current search space with their sub-spaces is to build a *pointer matrix*; each pointer refers to a child node (see Figure 6). Assume that the search space is partitioned along d dimensions simultaneously. The pointer matrix at each internal node then requires $O(2^{wd})$ storage, which precludes the choice of larger stride w. Existing algorithms, such as [1, 7], cut down the size of index pointer by choosing modest w for each node. Instead, we solve this problem by using *ID indexes* (see Figure 7) rather than the pointer matrix.

First we assigned each (aggregated) sub-space with a *space ID*. Then we create a 2^w-entry ID index for each dimension to map unit-spaces into the sub-spaces of each child nodes. By carefully designing the data structure, we can fix the size of each node in order to save the child nodes in continuous memory with a constant increment (node size) in memory address. Hence the memory address of corresponding child node can be directly obtained using the space ID and the address offset of the first child node. Because the space complexity of each internal node is reduced from $O(2^{wd})$ to $O(d*2^w)$, a greedy choice of w then makes sense.

4.3 Discussion

As a summary, we discuss the ideas and methods in *sBits* on the following aspects:

a. Stride: Fixed or Non-fixed? Because searching speed is the most important performance metric in high-end products, large and fixed stride ensures the worst case search time. Algorithms like HiCuts and HyperCuts can not use a large w, because the pointer matrix they use for packet search requires very large memory space accordingly. In addition, searching for an *optimized w* in HiCuts and HyperCuts is time-consuming. A fixed stride will also significantly reduce the pre-processing time.

b. Partition: Single-dimensional or Multidimensional? At each internal node, *sBits* suggest to partition the search space in multiple dimensions. Although a multi-step single dimensional partition seems to be able to perform more delicate optimization for partitions along each dimension, heuristics required for such optimization in all cases are hard to achieve. Moreover, even if such heuristics were obtained, they might be too complicated to be used due to the heavy computing burden. Thus *sBits* selects the most f discriminative dimensions to partition.

c. Aggregation: Adjacent or Non-adjacent? Non-adjacent unit-spaces may have same rules colliding with them, and hence according to projection-based partition, all these unit-spaces are assigned with same space ID. However, different from the exhaustive partition based on rule projection, the aggregated sub-spaces will be further partitioned. So the child nodes cannot handle non-adjacent sub-spaces. Even if the non-adjacent spaces can be converted to adjacent space by filling up the empty space with the same colliding rules, aggregating non-adjacent unit-spaces requires $O(N*2^{2w})$ time for each dimension, while adjacent aggregation needs only $O(N*2^w)$ time.

The original intention of our research is to combine the advantages of existing best algorithms. *sBits* can be viewed as a hybrid algorithm in terms of space partition and packet search. Globally, *sBits* partitions the search space with a decision tree structure [1, 7, 14]. Locally, at each internal node, the search step is implemented by table lookup [3, 4] using ID indexes.

5. Experimental Results

5.1 Rule Sets

We evaluate *sBits* both on real-life firewall and core router rule sets as well as on synthetic rule sets. The real-life rule sets are obtained from enterprise networks and major ISPs. Firewall rule sets are named FW1, FW2, FW3; Core router rule sets are named CR1, CR2, CR3, CR4; Synthetic rule set is SYN1, SYN2… SYN20. The largest real-life rule set (CR4) contains 1945 rules, and the largest synthetic rule set contains 2000 rules. All rules are 5-dimensional with 32-bit source and destination IP addresses represented as prefixes, 16-bit source and destination port numbers represented as ranges, and an 8-bit protocol.

5.2 Metrics

All the algorithms in our experiment are written in C codes and running in a PC with Pentium4 CPU. To test the performance of all the algorithms on both real-life and synthetic rule sets, we examine, for each rule set, the number of memory accesses as the *search time* (Time) and the amount of *memory usage* (Space) for the whole data structure built by the algorithms. Different from [7, 8] (where one memory access is a single 32-bit word access) one memory access here refers to reading a certain number (1~8) of continuous memory words. This is because today's most DRAM support burst mode reading, i.e. the time spent in reading continuous memory is very close to that of reading a single word.

5.3 Performance on Real-life Rule Sets

We first compare *sBits* with the algorithms of the best reported performance, including HiCuts, HyperCuts, RFC and HSM, on real-life rule sets. Due to patent issues, we were not able to obtain source codes from the authors and thus the codes of these algorithms are all written by ourselves. We make our best effort to make sure the fairness of our result analysis. Experimental results show that our codes achieved nearly the same performance compared to the experimental results reported in [7].

In comparison with algorithms using Trie-based partition, Table 1 and Table 2 compare *sBits* with HiCuts & HyperCuts respectively on spatial and temporal performance. We can see from these tables that, for all real-life rule sets, *sBits* achieves at least an order of improvement in memory usage, as well as a superior worst case query time.

To compare with the algorithms using Projection-based partition, Table 3 shows the memory usage of RFC, HSM and *sBits*. For the largest real-life rule set CR4, *sBits* uses 23 times less memory than HSM and 37 times less than RFC. Although the search speed of RFC and HSM are 50%~100% faster that of *sBits*, they require parallel searches in different fields, while *sBits* can be fully pipelined to implement for fast search.

Table 4 shows the full update time of typical heuristic algorithms in comparison with that of *sBits*. We see that even with the largest rule sets, *sBits* uses less than 1 second to construct the decision tree. In comparison, both RFC and HiCuts need tens of seconds in generating the whole data structure for search.

5.4 Performance on Synthetic Rule Sets

In order to compare their stability with large number of rules, we test *sBits* and HyperCuts on a series of synthetic rule sets. Although results on real-life rule sets are more persuasive, this task is made harder because the rule sets available to us is quite limited. Synthetic rule sets have size from 100 to 2000 and are named SYN# according to the number of rules (SYN1 has 100 rules and SYN20 has 2000 rules). We create these rule sets in the same way as [7], and all of these rules have identical distribution at each field.

Memory usage of HyperCuts and *sBits* on each synthetic rule set is depicted sequentially in Figure 8. We see that both HyperCuts and *sBits* perform stable with the number of rules less than 1700. But when the number of rules becomes larger (than 1700), the memory usage of HyperCuts has a sharp increase in comparison with that of *sBits*.

Similar results are obtained when we test other algorithms on these synthetic rule sets. *sBits* is proved to be more stable than all other algorithms we have tested. Such a conclusion is also supported by the results in Table 1 and Table 2 with real-life rule sets.

Table 1. Memory usage comparison:
sBits and HiCuts & HyperCuts (Unit: 32-bit word)

	No. Rules	HiCuts	HyperCuts	*sBits*
FW1	68	5,443	35,401	420
FW2	136	10,779	69,782	924
FW3	340	24,645	172,932	2,331
CR1	500	29,409	89,005	3,612
CR2	1000	979,736	871,541	28,287
CR3	1530	13,606,858	480,225	29,204
CR4	1945	5,928,724	672,442	43,183

Table 2. Worst case search time comparison:
sBits vs. HiCuts/HyperCuts. (Unit: Memory Access)

	No. Rules	HiCuts	HyperCuts	*sBits*
FW1	68	19	16	15
FW2	136	20	16	15
FW3	340	20	16	15
CR1	500	24	17	16
CR2	1000	30	17	16
CR3	1530	36	18	16
CR4	1945	34	18	17

Table 3. Memory usage comparison:
sBits vs. RFC & HSM (Unit: 32-bit word)

	No. Rules	RFC	HSM	*sBits*
FW1	68	200,652	10,223	420
FW2	136	209,602	27,657	924
FW3	340	296,382	65,581	2,331
CR1	500	264,987	29,814	3,612
CR2	1000	530,539	230,716	28,287
CR3	1530	863,476	486,857	29,204
CR4	1945	1,580,005	989,161	43,183

Table 4. Update time comparison:
sBits vs. RFC & HiCuts (Unit: milliseconds)

	No. Rules	RFC	HiCuts	*sBits*
FW1	68	1,492	73	1
FW2	136	1,762	211	22
FW3	340	3,185	465	40
CR1	500	4,597	409	56
CR2	1000	12,929	7661	261
CR3	1530	37,754	22,522	281
CR4	1945	67,087	21,248	350

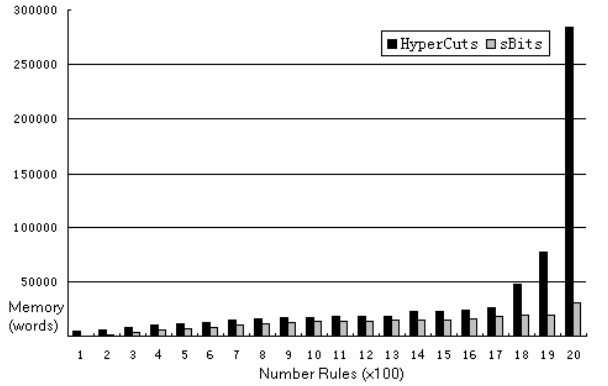

Figure 8. Stability: sBits vs. HyperCuts. (on synthetic rule sets SYN1, SYN2, …, SYN20)

6. Conclusion

Recall the incisive conclusion made by Pankaj Gupta in [1] that: "The theoretical bounds tell us that it is not possible to arrive at a practical worst case solution. Fortunately, we don't have to; No single algorithm will perform well for all cases. Hence a hybrid scheme might be able to combine the advantages of several different approaches." In this paper, we first suggest two generic procedures for packet classification problem, and then according to these two procedures we make a dissectional analysis of the existing algorithms to find their cohering relations. The algorithm *sBits* proposed in this paper is a hybrid scheme which uses a fixed-stride decision tree to partition the search space, along with an ID indexing data structure for packet search. Experimental results shows that *sBits* outperforms the best results of existing algorithms.

Future work can be conducted to introduce network traffic statistics into packet classification to dynamically optimize the decision tree structure and hence improve the average search speed. Future work also includes the implementation of *sBits* on new generation network processors. The codes we wrote for *sBits*, HiCuts, HyperCuts, RFC and HSM will be publicly available to encourage experimentation with classification algorithms.

Acknowledgements

This work is sponsored by the Intel IXA University Program.

References

[1] P. Gupta and N. McKeown, Packet classification using hierarchical intelligent cuttings, *Proc. Hot Interconnects,* 1999.

[2] M.H. Overmars and A.F. van der Stappen, Range searching and point location among fat objects, *Journal of Algorithms, 21(3)*, 1996, 629-656.

[3] P. Gupta and N. McKeown, Packet classification on multiple fields, *Proc. ACM SIGCOMM*, 1999, 147~160.

[4] B. Xu, D. Jiang, and J. Li, HSM: A fast packet classification algorithm, *Proc. 19th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, Taiwan, 2005, 1: 987-992.

[5] V. Srinivasan, G. Varghese, S. Suri and M. Waldvogel, Fast and scalable layer four switching, *Proc. ACM SIGCOMM*, 1998, 191-202.

[6] F. Baboescu and G. Varghese, Scalable packet classification, *Proc. ACM SIGCOMM*, 2001, 199-210.

[7] S. Singh, F. Baboescu, G. Varghese and J. Wang, Packet classification using multidimensional cutting. *Proc. ACM SIGCOMM*, 2003, 213-224.

[8] F. Baboescu, S. Singh and G. Varghese, Packet classification for core routers: Is there an alternative to CAMs? *Proc. IEEE INFOCOM*, 2003, 1:53-63.

[9] V.Srinivasan, S.Suri and G.Varghese, Packet classification using tuple space search, *Proc. ACM SIGCOMM*, 1999, 135-146.

[10] J. van Lunteren and T. Engbersen, Fast and scalable packet classification, *IEEE Journal on Selected Areas in Communications 21(4)*, 2003, 560-571.

[11] A. Feldman and S. Muthukrishnan. Tradeoffs for packet classification, *Proc. IEEE INFOCOM*, 2000, 3: 1193-1202.

[12] T. Lakshman and D. Stiliadis, High speed policy-based packet forwarding using efficient multi-dimensional range matching, *Proc. ACM SIGCOMM*, 1998, 203-214.

[13] T.Y.C Woo, A modular approach to packet classification: algorithms and results, *Proc. IEEE INFOCOM*, 2000, 3:1213-1222.

[14] F. Geraci, M. Pellegrini and P. Pisati, Packet classification via improved space decomposition Techniques, *Proc. IEEE INFOCOM*, 2005, 1:304-312.

[15] Y. Qi and J. Li, Dynamic cuttings: packet classification with network traffic statistics, *3rd Proc. International Trusted Internet Workshop*, 2004.

[16] P. Gupta and N. McKewon, Algorithms for packet classification, *IEEE Network 15(2)*, 2001, 24-32.

[17] D.E. Taylor, Survey and taxonomy of packet classification techniques, *ACM Computing Surveys 37(3)*, 2005, 238-275.

[18] M.E. Kounavis, A. Kumar, H. Vin, R. Yavatkar and A.T. Campbell, Directions in packet classification for network processors, *Proc. 2nd Workshop on Network Processors*, 2003.

[19] Y. Qi, B. Xu and J. Li, Performance evaluation and improvement of algorithmic approaches for packet classification, *Proc. International Conference on Network and Services*, 2005.

[20] D. Liu, B. Hua, X. Hu and X. Tang, High-performance packet classification algorithm for many-core and multithreaded network processor, *Proc. International Conference on Compiler, Architecture, and Synthesis for Embedded Systems (CASES)*, 2006, to appear.