A Fast Packet Classification by Using Enhanced Tuple Pruning

Pi-Chung Wang[1], Chia-Tai Chan[1], Wei-Chun Tseng[1], and Yaw-Chung Chen[2]

[1] Telecommunication Laboratories, Chunghwa Telecom Co., Ltd,
7F, No. 9 Lane 74 Hsin-Yi Rd. Sec. 4, Taipei, 106 Taiwan, R.O.C.,
Tel: +886-2-23265631, Fax: +886-2-23445700,
{abu,ctchan,wctseng}@cht.com.tw

[2] Department of Computer Science and Information Engineering,
National Chiao Tung University, Hsinchu, Taiwan, R.O.C.,
{ycchen}@csie.nctu.edu.tw

Abstract. In the packet classification, the route and resources allocated to a packet are determined by the destination address as well as other header fields of the packet such as source/destination address, TCP and UDP port numbers. It has been demonstrated that performing packet classification on a potentially large number of fields is difficult and has poor worst-case performance. In this work, we proposed an enhanced tuple pruning search algorithm called *"Tuple Pruning +"* that provides fast two-dimension packet classification. With reasonable extra filters added for *Information Marker*, only one hash access to the tuples is required. Through experiments, about 8 MB memory is required for 100K-filter database and 20 million packet per second (MPPS) is achievable. The results demonstrate that the proposed algorithm is suitable for high-speed packet classification.

1 Introduction

Traditionally, routers have forwarded packets based only on the destination address of the packet and do not provide service differentiation because they treat all traffic going in the same way. Increasingly, new services require more discriminating forwarding, called "Packet Classification". It allows service differentiation because the router can distinguish traffic based on source/destination address and application type. The process of mapping packets to different service classes is referred to as packet classification. The simplest, best-know form of packet classification is IP lookups, in which each rule specifies a destination prefix. The associated action is the IP address of next router that the packet must be forwarded. The other services which require packet classification include: access-control of firewalls, policy based routing, provision of differentiated qualities of services, and traffic billing, etc.

To describe the problem formally, we have to define the classifier and the filter. A classifier is a set of rules or filters that specifies the flows or classes. Packet classification is performed using a packet classifier, also called filter database. A

G. Carle and M. Zitterbart (Eds.): PfHSN 2002, LNCS 2334, pp. 180–191, 2002.
© Springer-Verlag Berlin Heidelberg 2002

filter F is called k tuple $F = (f[1], f[2], \ldots f[k])$ if the filter contents the k fields of the packet header, where each $f[i]$ is either a variable length prefix bit string, a range or a explicit value. A filter can be combined from many fields, for a packet header, the most common fields are the IP source address (SA), the destination address (DA), the protocol type and port numbers of source and destination applications and protocol flags. A packet P is said to match a particular filter F if for all i, the i_{th} field of the header satisfies the $f[i]$. Each filter has an associative action. For example, the filter F=(140.113.*, *, tcp, 23, *) specifies a rule that flows which address to the subnet 140.113 use the telnet application and the action of the rule may disallow these flows into its network. Beside of the action, the filter is usually given a cost value to define the priority in the database. The action of the least-cost matching filter will be used to process the arriving packet.

To perform packet classification on a potentially large number of filters on key header fields is difficult and has poor worst-case performance. In the previous work, tuple pruning search is proposed to achieve fast and scalable two-dimension (SA,DA) packet classification [1]. Through simulation, 11 hash accesses are required to finish a packet classification in the worst case as described in the literature. In this work, an enhanced tuple pruning search algorithm is proposed. With reasonable extra filters added, only one hash access to the tuples is required. Also, about 8 MB memory is required for 100,000-filter database. By using parallel hardware design, 20 million packet per second (MPPS) is achievable. The proposed algorithm is thus suitable for high speed packet classification.

The rest of the paper is organized as follows. Firstly, the related algorithms are introduced in Section 2. Section 3 presents the proposed algorithm. The experiment setup and results are presented in 4. Finally, a summary is given in Section 5.

2 Related Works

Recently several algorithms for packet classification have appeared in the literature [1], [2], [3], [4], [5], [6], [7], [8], [9]. It can be categorized into following classes: linear search/caching, hardware-based, grid of tries/cross-producting, recursive-flow classification, hierarchical intelligent cuttings, and hash-based solution. Many of these algorithms which provide fast lookup performance, required $O(N^k)$ memory space in the worst case, where N is number of filters and k is the number of classified fields. In the following, we briefly described the main properties of these algorithms.

Linear Search/Caching: The simplest approach to packet classification is to perform a linear search through all the filters. This requires $O(N)$ memory, but also takes $O(N)$ lookup time, which would be unacceptably large even for modest size filter sets. Caching is a technique often employed at either hardware or software level to improve performance of linear search. If packets from the same flow have identical headers, packet headers and corresponding classification solution can be cached. However, performance of caching is critically dependent

on having large number of packets in each flow. Also, if number of simultaneous flows becomes larger than cache size, performance degrades severely. Note that the average lookup time is adversely affected by even a small miss rate due to very high cost of linear search. Hence caching is much more useful when combined with a good classification algorithm that has a low miss penalty.

Hardware-Based Solutions: A high degree of parallelism can be implemented in hardware to gain speed-up advantage. Particularly, Ternary Content Addressable Memories (TCAM) can be used effectively for filter lookup. However,TCAM with particular word width cannot be used when flexibility in filter specification to accommodate larger filters is desired. It is difficult to manufacture TCAM with wide enough words to contain all bits in a filter. An interesting approach that relies on very wide memory bus is presented by Lakshamn *et al.* [4]. The scheme computes the best matching prefix for each of the k fields of the filter set. For each filter a pre-computed N-bit bitmap is maintained. The algorithm reads Nk bits from memory, corresponding to the best matching prefixes in each field and takes their intersection to find the set of matching filters. Memory requirement for this scheme is $O(N^2)$ and it requires reading Nk bits from memory. These hardware-oriented schemes rely on heavy parallelism, and requires significant hardware cost, not to mention that flexibility and scalability of hardware solutions is very limited.

Grid of Tries/Cross-Producting: For the case of 2-field filters, Srinivasan *et al.* presented a trie-based algorithm [3]. This algorithm has memory requirement $O(NW)$ and requires $2W-1$ memory accesses per filter lookup. A general mechanism called cross-producting is also presented. It involves performing best matching prefix lookups on individual fields, and using a pre-computed table for combining results of individual prefix lookups. However, this scheme suffers from a $O(N^k)$ memory blowup for k-field filters, including $k = 2$ field filters.

Recursive-Flow Classification: Gupta *et al.* presented an algorithm, which can be considered as a generalization of cross-producting [4]. After best matching prefix lookup has been performed, recursive flow classification algorithm performs cross-producting in a hierarchical manner. Thus k best matching prefix lookups and $k-1$ additional memory accesses are required per filter lookup. It is expected to provide significant improvement on an average, but it requires $O(N^k)$ memory in the worst case. Also, for the case of 2-field filters, this scheme is the same as cross-producting and hence has memory requirement of $O(N^2)$.

Hierarchical Intelligent Cuttings: Gupta *et al.* proposed a heuristic HICuts that makes hierarchical cuts in the search space [8]. It is difficult to characterize conditions under which such heuristics perform well, and the worst-case memory utilization for the HICuts scheme may explode.

3 Enhanced Tuple Pruning

3.1 Tuple Space Search

The tuple space idea generalizes the aforementioned approach to multi-dimensional filters [10]. A tuple T is defined as a combination of field length, and the

resulting set is called tuple space. Since each tuple has a known set of bits in each field, by concatenating these bits in order we can create a hash key, which can then be used to map filters of that tuple into a hash table. As an example, the two-dimensional filters F=(10*, 110*) and G=(11*, 001*) will both map to $T_{2,3}$. When searching $T_{2,3}$, a hash key is constructed by concatenating 2 bits of the source field with 3 bits of the destination field. Thus, the best matching filter can be found by probing each tuple alternately, and keeping track of the best matching filter. Since the number of tuples is generally much smaller than the number of filters, even a linear search of the tuple space results in a significant improvement over linear search of the filters.

To improve the speed of linear search, pre-computation and marker is used [1]. As a result, $2W - 1$ hash probes are required where W is the length of IP address. Another heuristic, tuple space pruning, performs lookups on individual fields to eliminate tuples that cannot match the query. Although this heuristic does not provide any improvement in the worst case, it performs well in the practical environment. Through the experiment, the number of probed tuples is reduced to about 10 in the worst case.

3.2 Enhance Mechanism: Tuple Pruning +

The tuple pruning search can be improved by adopting the concept of best match prefix (BMP). In the BMP problem, the longest matched prefix in the lookup procedure will be chosen to identify the route. Since there is only one longest matched prefix for each IP address, we can assign the filters related to the IP address to the tuple according to the longest matched prefix. Thus only the tuple with the BMP needs to be probed in the packet classification. Assume there are two filters in the two-dimension classifier: (10*,110*) and (1010*,110010*), as shown in Figure 1. These two filters will be assigned to the tuples according to their longest matched prefix, thus they will be located to $T_{2,3}$ and $T_{4,6}$, respectively. To further improve the tuple pruning search, an information marker is introduced to maintain the associated information for future tuple search.

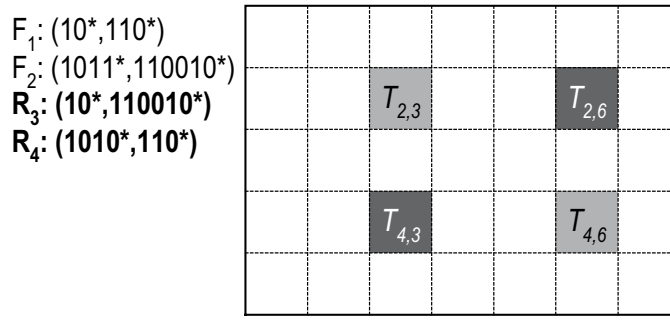

Fig. 1. A Sample Classifier with Two Filters.

By applying the idea in our proposed mechanism, one extra entry R_3 will be generated at $T_{2,6}$ and its action is equal to that of F_1. This filter will be referred for packets which SA is matched to 10* and DA is matched to 110010*, for example, the packet with header (100000,110010). According to the proposed idea, the lookup procedure will refer tuple (2,6), thus the associated action will be selected. We name this filter as a information-marker (i-marker here after). It is used to improve the search procedure as the marker used in [1], the major different is that i-marker is only put to the tuples with longer prefixes. The i-marker should also be added to $T_{4,3}$ and $T_{4,6}$. However, the one inserted into $T_{4,6}$ is identical to F_2, thus the action of F_1 will be compared with that of F_2. If the cost of F_1 action is lower, its action will occupy the action of F_2.

One of the major concerns about this approach is the number of the additional i-markers. Apparently, the number of the i-markers ties to the number of tuples with shorter prefix for each IP address. To illustrate this problem, we use the routing tables downloaded from [11], [12] as an example. In Figure 2, we show the number of shorter prefixes for each route prefix without counting the default route. Obviously, for most route prefixes, there are usually less than three shorter prefixes in the routing table and six in the worst case. On the other hand, at most 48 ($7^2 - 1$) extra filters might be generated for each inserted filter. However, the occurrence of the worst-case situation should be relatively low since only 5% of route prefixes have more than three and two shorter prefixes in the NLANR and the rest routing tables, respectively. And also, each shorter prefix may not appear in the classifiers. As a result, we believe that the extra cost should be acceptable with respect to the performance improvement.

Tuple Construction: To build the classification tuples, the procedure consists of two parts. First, each filter is inserted into the associated tuple according to its length. In the mean time, a prefix tree should be constructed to record the referred prefixes in the filters. A binary tree or the multi-bit tree proposed in

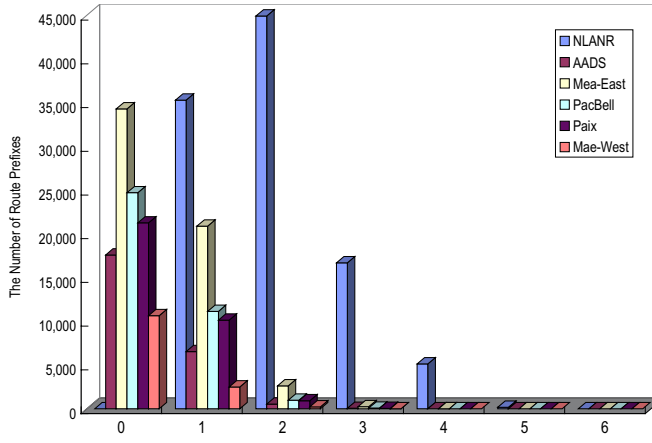

Fig. 2. The Number of Shorter Prefixes for Each Route Prefix.

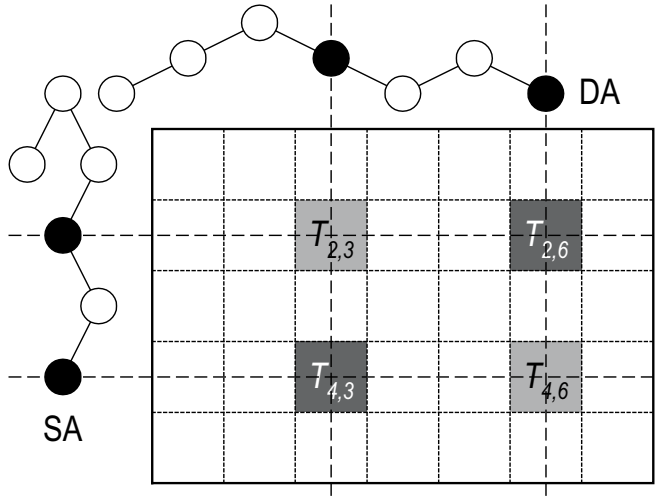

Fig. 3. The Tuple Construction from the Prefix Tree.

[13] can be used as the prefix tree. Then, the pruning table for each dimension will be generated from the prefix tree. Also, it will keep track of the relationship between each route prefix and its shorter prefixes, as shown in Figure 3. Second, the i-marker is added to the tuples from the prefix tree. For example, if there are m longer prefixes for a given SA and n longer prefixes for a given DA, then there are at most $m \times n$ i-markers will be added to the tuples. Before the i-marker is inserted into the tuple, the existence of duplicate filter is checked. If there is no duplicate entry, the i-marker is inserted. Otherwise, the cost of filter actions will be compared for the tuples with duplicate filter and the lower-cost action will be recorded in the entry. For the sake of ease update, each entry in the tuple should have two action fields: one is the lowest-cost action related to the filter and another is its original action.

Note that the insertion of i-markers will not affect the construction of pruning table since it is based on the original filters. Furthermore, the filters with at least one wildcard will not be inserted into the tuples. Those filters only need to be inserted into the prefix tree since they can be treated as single dimension prefixes. Thus it can reduce the number of i-markers.

Search: The classification procedure consists of two lookups of pruning tables and one hash lookup to the tuple. Firstly, the BMP lookup is performed in pruning tables for each dimension. However, the lookup result fetched here is different from that of IP route lookup, in which only the length of the BMP is needed in the pruning lookup. After the length of two BMPs l_1 and l_2 are found, the tuple (l_1, l_2) will be probed for the matched filter. Obviously, the tuple space lookup performance mainly ties to the lookup performance of pruning tables. The fast lookup algorithm proposed in the previous schemes can be applied to provide good performance.

Update: The update procedure is a little complex due to the pre-computation. However, the re-construction is not required in the proposed scheme. The table update can be divided into: change of filter action, insertion and deletion of filter. We only explain how to perform filter insertion and deletion. The change of filter action is trivial since it can be treated as re-insert the filter with new action.

To deal with the inserted filter, the prefix tree for each dimension is maintained. Firstly, it will be inserted into the prefix tree, and then the longer prefixes for the SA and DA are found from the prefix trees. With the lengths of the longer prefixes, the set of tuples which are covered by the inserted filter is calculated. An i-marker is inserted into each tuple in the set. If there is a filter with the same key as the i-marker, a cost comparison is performed and the action with lower cost will be left in the entry. Furthermore, the tuples covered by the lower-cost filter will not be probed for the insertion since they will not be affected by the inserted filter, as shown in Figure 4. A filter F_5 is inserted into the two-filter database of Figure 1. After inserting the SA and DA into the prefix trees, the set of probed tuples are derived. According to the row-major order, the i-markers are put into $T_{1,3}$, $T_{1,6}$, $T_{2,2}$ and $T_{4,2}$, respectively. While traversing $T_{2,3}$, a collision with F_1 is encountered. After comparing their cost, if the cost of F_5 is lower, its action will replace the lowest-cost action field of F_1 and keep traversing the remaining tuples. Otherwise, the entry in $T_{2,3}$ will remain unchanged and the remained three tuples ($T_{2,6}$, $T_{4,3}$ and $T_{4,6}$) which are covered by F_1 will not be probed in this insertion. In the worst case, it will update W^2 tuples.

The procedure of filter deletion is a little similar to that of filter insertion. Now we use the filter database with newly inserted F_5 in Figure 5 as an example. If the filter F_1 is deleted from the database, the tuples covered by it will be probed for possible update. However, before the tuple-probe proceeding, the nearest filter which covers F_5 should be found for possible referring. This is because if there are probed tuples with i-markers or filters with cost higher than F_1 and F_5, the action of F_5 should replace those entries to ensure that the lowest cost

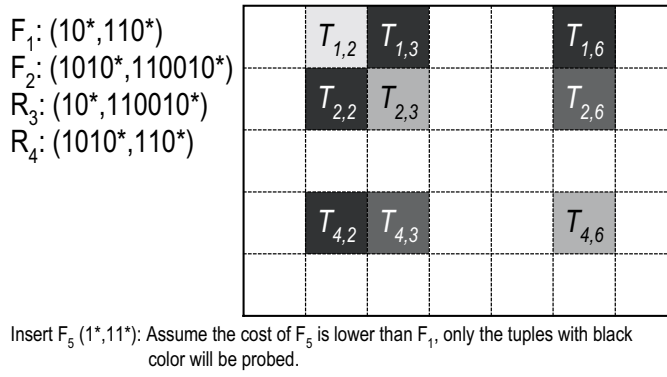

Fig. 4. An Example of Filter Insertion.

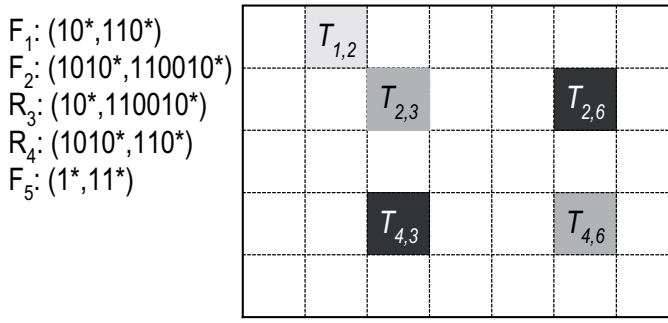

Fig. 5. An Example of Filter Deletion.

action will be taken. Consequently, the i-markers in the probed tuples will be refreshed by F_5. Also, the filters will be checked whether the cost is higher than F_5. If yes, they will be occupied by the action of F_5. Otherwise, their actions will be used instead. The time complexity for update is also $O(W^2)$.

Implementation: The pruning tuple space search can be implemented with software or hardware. With software implementation, the total lookup time is $(lookup(SA) + look(DA))$ plus one hash access time. To deal with the potential large number of entries, the hash function can allocate multiple entries in a pool to fit the cache line.

The lookup performance can be further improved through hardware implementation. By exploiting hardware parallelism, the total lookup time of the pruning tables is reduced to $max(lookup(SA), lookup(DA))$, as shown in Figure 6. It can also perform the pruning and hash simultaneously by adopting pipeline design to achieve higher throughput. As a result, we can accomplish one packet classification within *maximum(pruning(SA), pruning(DA), one hash access to the tuple).*

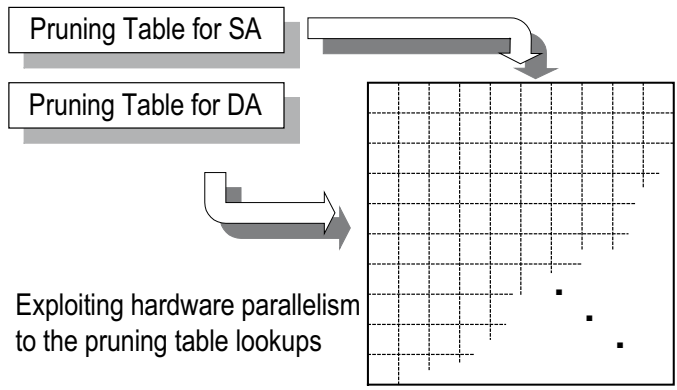

Fig. 6. Implement with Parallel Hardware.

Since the most memory accesses refer to the pruning, we can use high-speed SRAM as its storage. By adopting the existing IP lookup algorithms for pruning, such as multibit trie [13] or multiway search tree [14], the pruning time can be reduced to less than 50 ns easily (less than ten memory accesses). Assume that one hash access time without collision is 50 ns (one 50-ns DRAM access time), thus the proposed scheme can achieve 20 MPPS.

4 Performance Evaluation

To evaluate the performance of the proposed scheme, we use the randomly generated filter database with 5K to 1M entries. This is mainly because that the filter database is usually considered as secret data commercially. Also, most of them are relatively small, such as the filter databases used in [4]. Thus we generate the filter database from the routing table in NLANR. There are 102,309 prefixes in the sample routing table [11]. We use two different sampling schemes to generate the (SA,DA) filters: the first one is to choose the prefixes uniformly [1] and the other is to concentrate 80% filters in 20% address space to show the locality [9]. Note that the filters with wildcard are not considered in the simulation because they will be inserted into the pruning table and will not affect the tuples. The filter length distribution with 100,000 filters with 80% locality is shown in the right part of Figure 7 which is similar to the figure of the uniformly chosen filters. Obviously, most filters correspond to the tuples near (24,24). A darker color indicates that there are more filters in the tuple.

We first examine the filters database with 80% locality. The major performance metrics ties to the number of i-markers. Since the size of i-marker is equal to the filter, we use the term "entry" to cover both. From Table 1, we can see that the numbers of entries are about three to six times of the original tables. However, with a larger database (larger than 10,000 entries), the increased entry ratio is lower with respect to the smaller database (1,000). This is because with more entries in the table, the probability to generate an i-marker with collision filter is

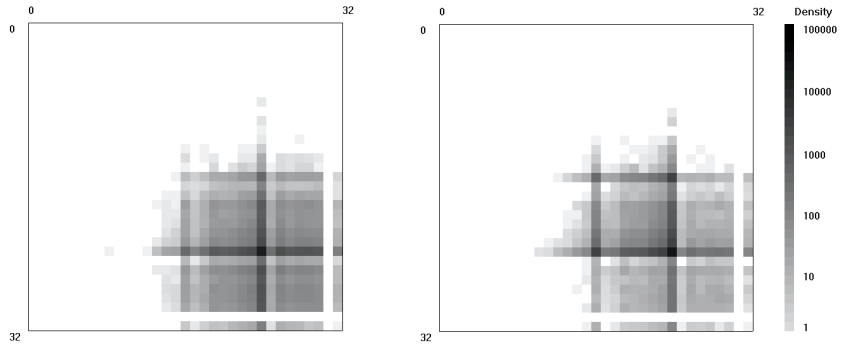

Fig. 7. The Filter Lengths Distribution of Original Database. (Left:Random, Right:80% Locality)

Table 1. The Number of Entries versus the Pruned Tuples without i-Marker. (80% Locality)

Filter Count	Original Scheme			Proposed Scheme		
	Tuples	Probes	Entry Count	Tuples	Probes	Entry Count
1,000	129	8	1,000	153	1	5,623
5,000	224	11	5,000	249	1	14,105
10,000	295	11	10,000	318	1	30,654
50,000	365	15	50,000	377	1	162,931
100,000	353	16	100,000	360	1	293,217
500,000	442	30	500,000	462	1	2,400,103
1,000,000	504	51	1,000,000	530	1	5,628,952

also higher, this reduces the ratio of increased entries. It does suggest good scalability, especially under the speed-critical environment, the proposed scheme has apparent improvement. The result for the random-generated database is shown in Table 2. The number of pruned tuple is slightly reduced because the address locality may result in more intersection in the pruning. However, the number of entries is increased for the large database (for database with more than 50,000 filters); this is because that large amount filters result in more related prefixes in the database. However, without i-marker, the probed tuples will increased to 51 in the worst case for 1M-filter database, i.e., at least 51 memory accesses. Obviously, the speed improvement is necessary, even with about 6 times storage.

For the random-generated database, the number of generated entries is more than that in the 80%-locality database. This is because the wide-spread filters might cause more i-markers in the worst case. We assume that the memory utilization of the hash table is 50%. Thus, for the 100k-filter database, it requires about 8 MB memory, whose cost is lower than US$ 10. However, it can achieve about 20 MPPS by using the 50-ns DRAM. While the database enlarges to 1M filter, it will requires about 130 MB memory without speed degradation.

Table 2. The Number of Entries versus the Pruned Tuples without i-Marker. (Random)

Filter Count	Original Scheme			Proposed Scheme		
	Tuples	Probes	Entry Count	Tuples	Probes	Entry Count
1,000	139	2	1,000	140	1	2,009
5,000	242	4	5,000	246	1	12,438
10,000	274	5	10,000	276	1	23,583
50,000	334	12	50,000	341	1	195,990
100,000	361	14	100,000	375	1	374,718
500,000	440	31	500,000	459	1	2,685,592
1,000,000	468	41	1,000,000	491	1	6,814,934

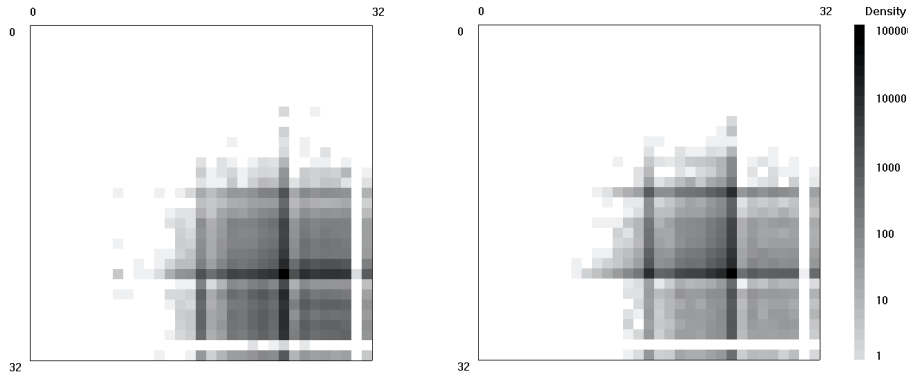

Fig. 8. The Filter Lengths Distribution of Database with i-Markers. (Left:Random, Right:80% Locality)

The filter lengths distribution graph from Figure 7 is shown in Figure 8. One can see that the number of required tuples is increased and the colors of most of the blocks are darker than that in the previous graph. Furthermore, the number of colored blocks is increased because the i-markers might be inserted to the tuples without filter originally.

5 Conclusion

In this paper, we propose a remarkable enhancement to the previous work. By using the pre-computation and i-markers, we can reduce the number of probed tuples from the worst-case $O(W^2)$ to $O(1)$. The incremental update is also supported. In the worst case, the number of generated i-markers are four times of the original filters for the 100,000-filter datebase. From the simulation, the proposed scheme with parallel design can achieve 20 MPPS in the worst case. In the future, we will focus on dynamic space-cutting algorithm to further reduce the required storage.

References

1. V. Srinivasan, G. Varghese and S. Suri: Packet Classification using Tuple Space Search. ACM SIGCOMM. (1999) 135–146
2. T.V. Lakshman and D. Stidialis: High Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching. ACM SIGCOMM. (1999) 203–214
3. V. Srinivasan, G. Varghese, S. Suri and M. Waldvogel: Fast Scalable Level Four Switching. ACM SIGCOMM. (1998) 191–202
4. Pankaj Gupta and Nick McKeown: Packet Classification on Multiple Fields. ACM SIGCOMM. (1999) 147–160
5. Anja Feldmann and S. Muthukrishnan: Tradeoffs for Packet Classification. IEEE INFOCOM. (2000) 1193–1202

6. Thomas Woo: A Modular Approach to Packet Classification: Algorithms and Results. IEEE INFOCOM. (2000) 1213–1222
7. M. Buddhikot, S. Suri and M. Waldvogel: Space Decomposition Techniques for Fast Layer-4 Switching. IFIP Sixth International Workshop on High Speed Networks. (2000)
8. Pankaj Gupta and Nick McKeown: Packet Classification using Hierarchical Intelligent Cuttings. Hot Interconnects VII. (1999)
9. Ying-Dar Lin, Huan-Yun Wei and Kuo-Jui Wu: Ordered lookup with bypass matching for scalable per-flow classification in layer 4 routers. Computer Communications, Vol. 24. (2001) 667–676
10. M. Waldvogel, G. Varghese, J. Turner and B. Plattner: Scalable High Speed IP Routing Lookups. ACM SIGCOMM. (1997) 25–36
11. NLANR Project: National Laboratory for Applied Network Research. See *http://www.nlanr.net*
12. Merit Networks Inc.: Internet Performance Measurement and Analysis (IPMA) Statistics and Daily Reports. IMPA Project. See *http://www.merit.edu/ipma/routing_table/*
13. V. Srinivasan and G. Varghese: Fast IP lookups using controlled prefix expansion. ACM Trans. On Computers, Vol. 17. (1999) 1–40
14. P.C. Wang, C.T. Chan and Y.C. Chen: Performance Enhancement of IP forwarding by using Routing Interval. Journal of Communications and Networks, Vol. 3. (2001) 374–382.