Optimal Tuple Reduction for Fast Two-Dimension Packet Classification

Pi-Chung Wang*, Chia-Tai Chan, Wei-Chun Tseng and Yaw-Chung Chen

Abstract

Packet classification categorizes packets into flows based on some predefined filters. Nowadays the packet classification techniques play an important role for many new Internet services. Rectangle search is a well-known packet classification scheme which is based on multiple hash accesses for different filter length. It shows good scalability with respect to the number of filters; however, the lookup performance is not fast enough. For example, through experiments, each packet classification takes about 40 hash accesses in a 100,000-filter database and each hash access may take more than one memory access. Obviously, this is not capable to provide gigabits throughput.

In this paper, we propose an efficient **Tuple Reduction Algorithm** to improve the rectangle search. The **Tuple Reduction Algorithm** is based on the filter duplication. In spite of the increased number of filters, the pre-computation information is dramatically reduced, the performance has increased two times while only about one quarter storage is required. The experimental results indicate that the proposed scheme can fulfill OC-48 throughput.

Keywords

Internet, High-Speed Network, Packet Classification.

The authors are now with the Telecommunication Laboratories, Chunghwa Telecom Co., Ltd, 7F, No. 11 Lane 74 Hsin-Yi Rd. Sec. 4, Taipei, Taiwan, R.O.C. (TEL: +886-2-23265631, FAX: +886-2-23445700, e-mail: {abu, ctchan, wct-seng}@cht.com.tw).

Yaw-Chung Chen is now with the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C. (e-mail: ycchen@csie.nctu.edu.tw).

I. INTRODUCTION

Nowadays, the bandwidth bottleneck ties to the Internet lookup which is performed in the router. The Internet lookup, including IP lookup and packet classification, is used to decide the action for each incoming packet. Hashing is a widely used method to perform fast lookup. Several hash-based schemes have been proposed to solve Internet lookup problem [1], [2]. Rectangle search [1] is a well-known packet classification scheme which is based on multiple hash accesses for different filter length. It was proposed to show the lower bound $O(2W-1)$ of searching for a lowest-cost matching filter, where W is the length of the IP address. It shows good scalability with respect to the number of filters; however, the lookup performance is not fast enough. For example, through experiments, each packet classification takes about 40 hash accesses in a 100,000-filter database and each hash access may take more than one memory access. Therefore, the performance is not capable to provide gigabits throughput.

In this paper, we propose an efficient **Tuple Reduction Algorithm** to improve the rectangle search. The proposed **Tuple Reduction Algorithm** is based on filter duplication. The near-optimal algorithm are presented to lessen number of tuples. The number of filters is increased; however, the resulting pre-computation information required in the rectangle search is dramatically reduced. Through experiments, we show that it only uses about quarter storage and increases throughput about two times. Furthermore, the proposed scheme can fulfill OC-48 throughput.

The rest of the paper is organized as follows. The previous works are introduced in Section II. Section III presents the proposed scheme. The experiment setup and results are shown in IV. Finally, a summary is given in Section V.

II. PREVIOUS WORKS

Several schemes that work well in practice have been proposed [3], [1], [4], [5]. The bit-level parallelism scheme proposed in [5] and the Crossproducting [4] take O(N^2) memory. The Recursive Flow Classification algorithm presents good lookup results and a good hardware implementation with moderate memory requirements for real life databases with O(N^2) memory in the worst case. An algorithm called Hierarchical Intelligent Cuttings was proposed in [6] which performs packet classification at high speeds with affordable memory utilization. A scaled model of the filter databases based on the current databases was presented in [7]. It shown that those algorithms which conform to the model perform packet classification well. While the above discussion focuses on algorithmic approaches, Ternary Content Addressable Memory (TCAM) technology has advanced significantly in the packet classification. However, it does not scale well. Algorithms that use conventional Static Random Access Memory (SRAM) can outperform TCAMs when large databases have to be supported. Research in scalable algorithmic approaches is still considered important.

The tuple space search is one of the packet classification algorithms. A tuple is a collection of filters with specific prefix length. For example, the two-dimension filters $F = (10*, 110*)$ and $G = (11*, 001*)$ will both map to tuple $T[2, 3]$. For the sake of efficiency, the search within a tuple is performed by hashing since the key lengths are identical. Thus each tuple access indicates one hash access which might need multiple memory accesses. The resulting set of tuples is called as *tuple space* (**TS**). When searching for a best matching filter, it can examine all the tuples in the tuple space. It is observed that the number of tuples in a real database is much smaller than the number of filters. For a large filter database, the scale of the tuple space is unlikely to grow beyond few hundreds. This is because the most databases use only few prefix

lengths corresponding to CIDR.

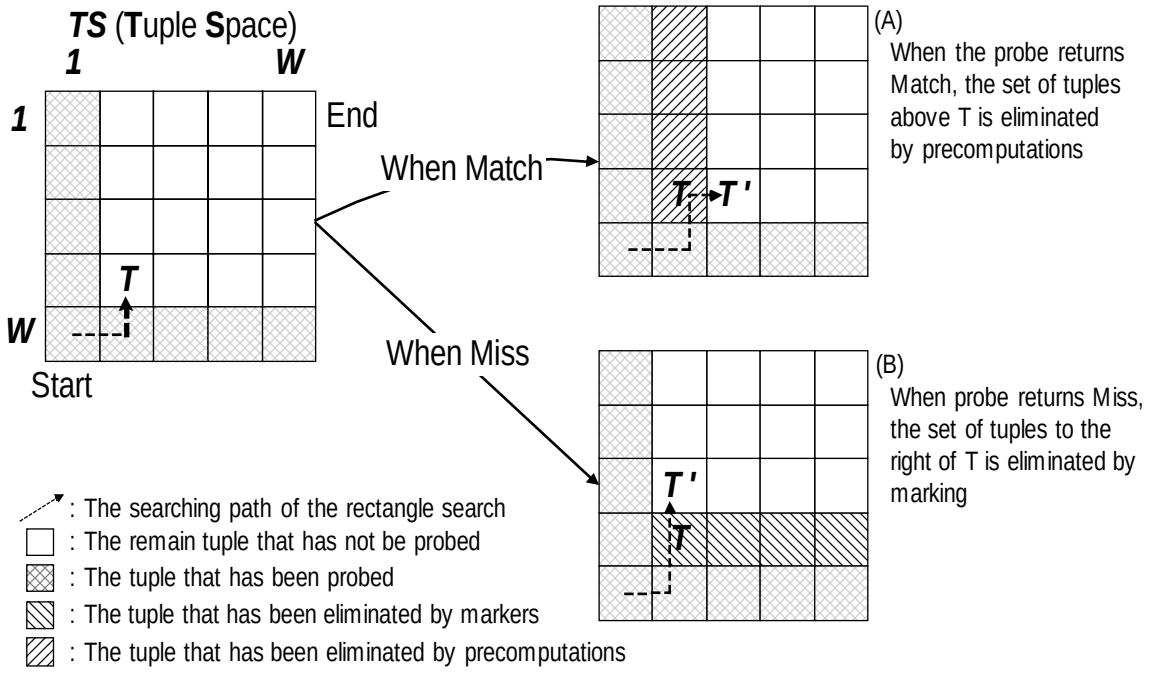


Fig. 1. The Rectangle Search Algorithm.

In [1], the rectangle search was proposed to further improve the lower bound of tuple lookup. It was proved that the lower bound is O$(2W - 1)$ for a O$(W \times W)$ rectangular tuple space, where W is the number of distinct prefix lengths. The main idea is using each tuple probing to eliminate a set of the tuples which is shown in Figure 1. While probing tuple T, the tuples above T would be eliminated if it returns a **"Match"**. Otherwise, the tuples in the right side of tuple T would be discarded. To archive this goal, the markers and pre-computation mechanism are required. Assume the number of filters is N, the rectangle search requires O(NW) memory space. The detail information about the markers will be introduced in the next section. Note that if the number of distinct lengths reduces, the lookup performance will increase as well. Also, the number of generated markers could be eliminated under certain conditions.

III. TUPLE REDUCTION ALGORITHM

As described above, pre-computation and markers are used to perform rectangle search. In Figure 2, it shows the relationship between the filters and markers. The filter database contains 203 filters which occupy across 5 tuples. Each filter has to generate one marker to each left-side tuple for indicating that there exists a potential match with longer length. Accordingly, 703 markers will be generated and result in 906 entries totally. Due to the unbalanced length distribution of routing prefixes, it is highly possible to incur a size explosion in the filter databases. Consequently, we will presents how to lessen the number of tuples to advance the native rectangle search.

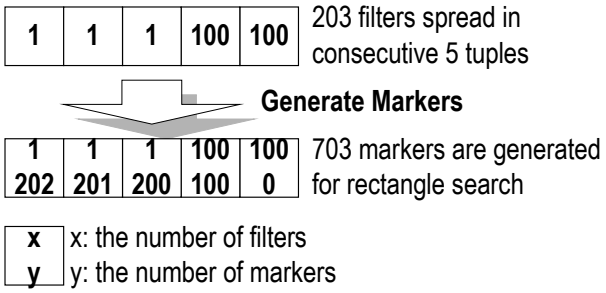


Fig. 2. The Relationship Between Filters and Markers.

A. Filter Expansion

The lookup performance of the rectangle search can be improved by reducing the distinct lengths of both dimensions. To achieve this, a well-known skill based on entry duplication can be used. For example, as described in [8], the authors improve the lookup performance by lessening the distinct prefix lengths. To minimize the increasing storage resulted from the entry duplication, the dynamic programming is deployed. The other algorithms [9], [10], [11] are based on the similar mechanism but with different compression schemes. Briefly, the algorithms

in this category either need larger storage or complex compression logic as tradeoff to achieve higher throughput.

By applying the concept to the rectangle search, it is possible to obtain faster lookup while retains close or even smaller storage. This is because that the tuple reduction results in fewer generated markers. We use the example in Figure 2 to explain. As shown in Figure 3, the three shorter filters are expanded to the fourth tuple which corresponds to longer filter. The required markers are reduced dramatically since the lookup will start from the fourth tuple which contains the duplicate filters of original ones.

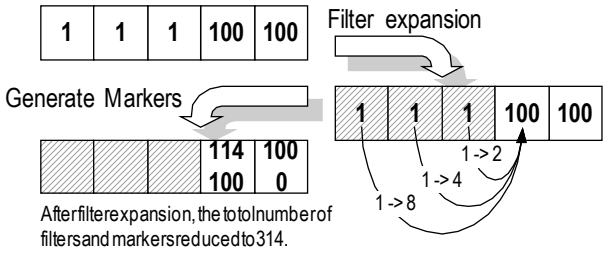


Fig. 3. The Tuple Space Search with the Filter Expansion.

We further illustrate the filter expansion with two dimension in Figure 4. The filter $f(11*, 10*)$ at the tuple $T_{2,2}$ will be expanded to the destination tuple $T_{3,3}$. Firstly, the source prefix of filter is expanded from 11* to the set of prefixes 110* and 111*. With the same procedure, the destination prefix of filter is expanded from 10* to 100* and 101*. After expanding both the source and destination prefix, we cross-product 2 set of prefixes and get a set of new filters $f_1(110*, 100*)$, $f_2(111*, 100*)$, $f_3(110*, 101*)$ and $f_4(111*, 101*)$, which is equal to the original filter $f(11*, 10*)$.

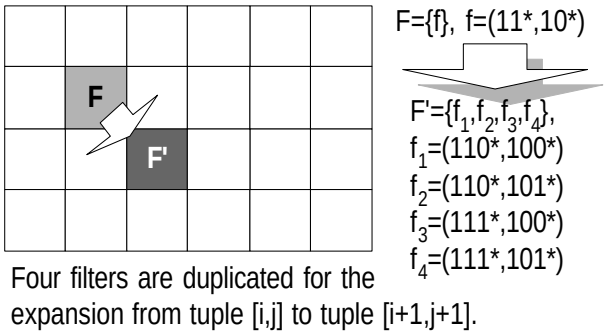


Fig. 4. The Filter Expansion.

B. Dynamic Programming

To obtain the optimal filter expansion, the dynamic programming can be applied to minimize the cost of filter expansion. To determine the filter expansion in the tuple space, the original filters, the duplicate filters and the markers should be considered. Figure 5 illustrates the cost for a filter expansion. Several decisive factors should be included. The first one is the duplicated filters from the expansion as described above. The second one is the number of generated markers in the source tuple. These markers will be eliminated due to the tuple expansion. Consequently, the number of markers for the duplicated filters is counted. The minimal expansion cost can be formulated as a recursive equation, as listed below. The cost will be derived to determine whether the expansion is effective.

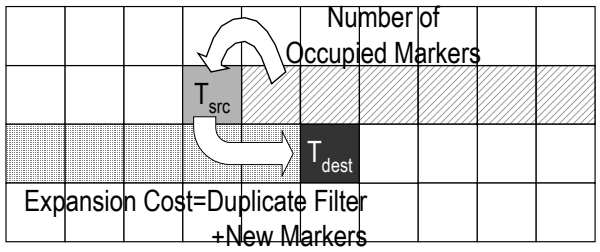


Fig. 5. The Dynamic Programming for the Filter Expansion.

$$
\begin{aligned}
DynaP(TS) = \; & \min_{i=1}^{|TS|}\{DynaP(TS - T_i) \\
& + \min_{j=1}^{|TS|}\{Expansion(T_i, T_j) \\
& - Sum\_of\_Filters(T_i)\}\};
\end{aligned} \tag{1}
$$

Where

$$
\begin{aligned}
Expansion(T_{src}, T_{dest}) &= 2^{Dist(T_{src}, T_{dest})} \times |T_{src}| \\
&\quad \times Left\_Tuples(T_{dest}); && (2) \\
Sum\_Of\_Filters(T_{R,C}) &= \sum_{col=C}^{C \leq W} |T_{R,col}|; && (3) \\
Left\_Tuples(T) &= Number\,of\,tuples \\
&\quad in\,the\,left\,side\,of\,T; && (4)
\end{aligned}
$$

Though the optimal solution can be derived, the main drawback of the dynamic programming is its time complexity. The formula can be expressed as $f(m) = mf(m-1) + m^2$, where m is the number of the tuples and m^2 is the complexity of determining where a tuple is expanded to. Thus it results in a exponential cost-function $f(m) \leq 2^m$.

C. Semi-Optimization Algorithm

In the previous subsection, we describe the dynamic programming scheme to derive the optimal expansion. However, the complexity is too high to be practical. Consequently, we propose a heuristic semi-optimization algorithm by restricting the expansion. In the new algorithm, the tuples are only expanded to those with more filters. This is because the expansion cost for those tuples with more filters are usually higher. Thus we start the expansion from the tuples with

least filter. The expansion combinations can be significantly reduced to $M \times M$ where M is the number of tuples. We further restrict the maximum expansion distance. For each tuples, the expansion is allowed within a predefine region, such as a 3×3 region from the source tuple $T_{x,y}$ to $T_{x+2,y+2}$. The semi-optimization algorithm is listed below. The tuples are sorted by the number of filters. In the iteration i $(1 \leq i \leq M)$, $Cost\_Old$ indicates the current cost of T_i without expansion. T_{dest} is the best expansion tuple for tuple T_i within the restricted region. $Cost\_New$ indicates the cost after expanding tuple T_i to T_{dest}. If the $Cost\_New$ is greater, it does nothing and continues to process the next iteration. Otherwise, the filters in T_i are expanded to T_{dest}.

Semi-Optimization Algorithm

[Functions]

$Markers(T)$: Generated markers for the filters in tuple T.

$Optimal\_Expansion(T)$: Optimal expansion for tuple T

within the restricted region.

[Input] Ascending-ordered Tuples $\{T_1, T_2, \ldots, T_M\}$

[Output] The Optimized Tuples

For ($i = 1; i \leq M; i++$) {

$Cost\_Old = |T_i| + Markers(T_i) + Sum\_of\_Filters(T_i)$;

$T_{dest} = Optimal\_Expansion(T_i)$;

$Cost\_New = Expansion(T_i, T_{dest})$;

If $(Cost\_Old \geq Cost\_New)$

Expand Tuple T_i to T_{dest};

}

D. Tuple Reduction in Sparse Area

In the section III-C, we eliminate the required storage for the rectangle search to a near-optimum level. For the sake of higher lookup speed, we present how to eliminate the number of tuples in advance by using reasonable storage as a tradeoff. In the Internet, the length of the routing prefixes are always unbalanced-distribution. Thus most existing IP routing lookup scheme have to deal with this. Although the large filter databases are not commonly available yet, we believe that the unbalanced-distribution will sustain. We use the random generated filter database with 100K filters to examine the affection. As shown in Figure 6, we found that the most filters are located on the region from the up-left tuple $T_{16,16}$ to the bottom-right tuple $T_{24,24}$. This is because the routing prefixes are mainly with length 16-24 bits [2]. By separatin the **TS** into nine regions, one can see that only fraction of tuples are occupied by few filters in region **A,B,C,D,G**. Though the most tuples in the region **F,H,I** are occupied, the number of filters is relatively sparse as compared to the tuples in region **E**. These tuples with few filters are the main obstacle to the search speed since even with only one filter, one tuple access is still required.

To deal with this, we propose an enhancement to collect filters within the sparse regions. We set a value *MAX_FILTERS* and choose a target tuple in the region. Consequently, the nearest non-empty tuple is expanded to the target tuple. If the number of the filters in the target tuple does not exceed *MAX_FILTERS*, the procedure will repeat. Otherwise, another target tuple in this region are chosen and perform the procedure again.

IV. Performance Evaluation

In this section, we show the results of the experiments to demonstrate that the tuple reduction algorithm can improve both the lookup performance and the memory requirement. To evalu-

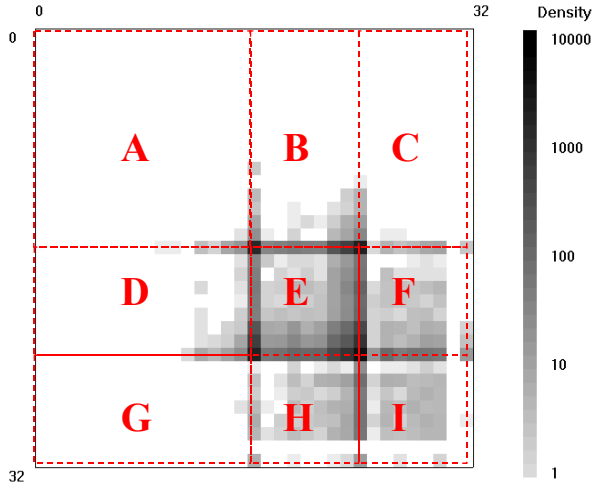
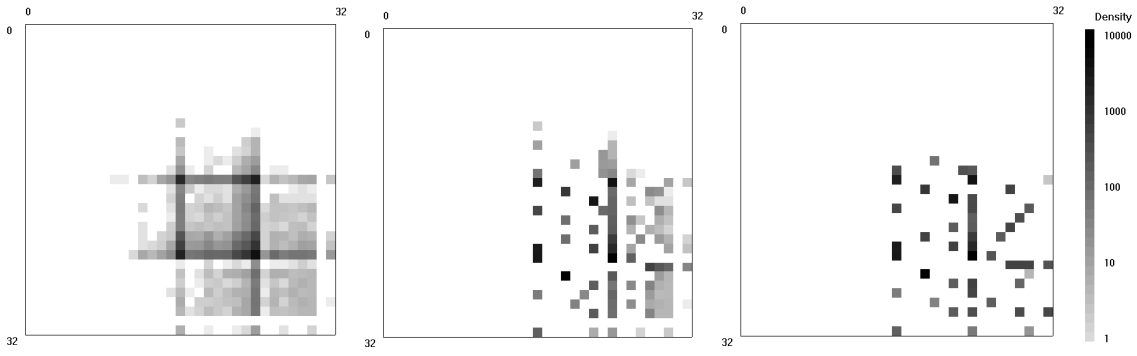


Fig. 6. The Unbalanced Filter Distribution.

ate the performance of the proposed scheme, we generate the filter databases from the routing table downloaded from the NLANR [12], which contains 102,309 prefixes with 30 next-hops. This is because the real filter database is usually considered as secret data in commerce. Ten ⟨source prefix, destination prefix⟩ filter databases are produced by sampling the routing prefixes randomly. The minimal database contains 10,000 filters, and the size of database increases by 10,000 filters until it reaches 100,000 filters.

In Figure 7, it shows the filter distributions in the **TS** for the 100K-filter database. After applying the semi-optimization algorithm, the remaining tuples are decreased to 85, as shown in Figure 7(b). But there still remains many tuples which contain few filters (< 100 filters). In the other words, it takes half memory accesses to probe these 20% filters in the **TS**. By using the tuple reduction for the sparse region, the number of the tuples can be reduced dramatically, as shown in Figure 7(c). The number of occupied tuples is reduced to 45, which shows an improvement with factor of eight.



(a) The distribution of the original database (350 tuples). It is easy to see that there are many tuple with few filters and the performance of searching is poor.

(b) After the semi-optimization, the number of tuples is significatly reduced, but there are still some tuples with few filters.

(c) After collecting the filters on the sparse regions, the number of the filters is usually reduced below 50.

Fig. 7. The Filter Distribution after the Filter Expansion.

Consequently, we use the proposed scheme as the combination of the semi-optimization algorithm and the sparse-region reduction. Figure 8 shows the number of the filters and the generated markers. It is easy to see that the proposed scheme outperforms the native rectangle search since the number of markers is significantly reduced. The proposed scheme only require about one quarter storage as compared to the native rectangle search.

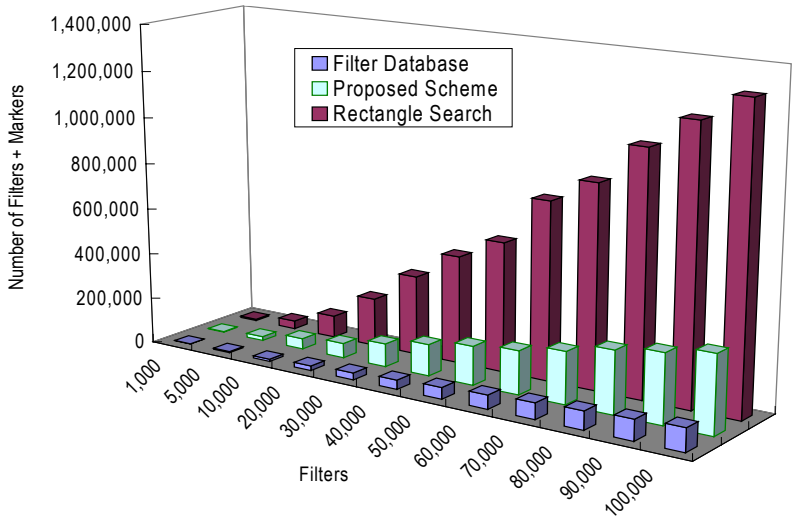


Fig. 8. The Memory Requirement.

To demonstrate the lookup speed of the proposed scheme, we generate 10 million packets randomly and record the number of the hash probes. The search performance is shown in Figure 9. The proposed scheme probes about half tuples than the native rectangle search to find a lowest-cost filter. Furthermore, we can find that the curves in the figure is very stable. This is because that the number of the tuples is not proportional to the number of filters.

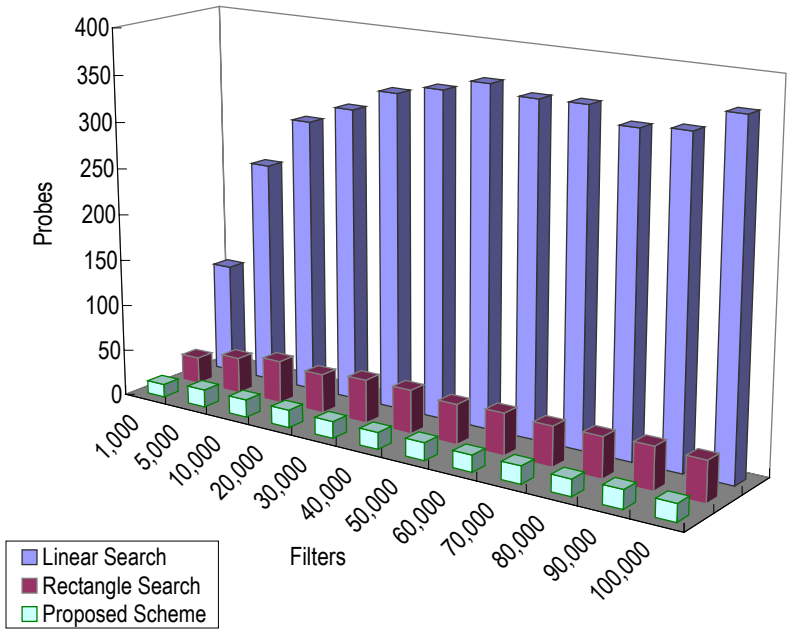


Fig. 9. The Search Performance.

Table I shows the detailed results of the examination. The memory requirements are derived by using the 80-bit filter. (32 bits source prefix, 32 bits destination prefix, 8 bits cost and 8 bits for the nexthop.) For the proposed scheme, it can achieve about twice throughput as the rectangle search while only quarter storage is required. By using the commercial available 30-ns DRAM, the proposed scheme can accomplish one lookup within 570 ns, i.e., provide OC-48 throughput with 256-byte average packet size.

TABLE I

PERFORMANCE COMPARISONS.

	Filter Count	Rectangle Search	Tuple Reduction Algorithm
Entries	100,000	1,288,823	335,882
Tuples	366	366	55
Max Probes	NA	59	26
Min Probes	NA	28	15
Avg Probes	NA	44	19
Size (KBytes)	976	12,586	3,280

V. CONCLUSION

In this work, we propose a near-optimal scheme to improve the rectangle search algorithm in both performance and the storage. We address how the filter expansion can be applied to the rectangle search to reduce the number of markers and tuples. Due to the high cost of dynamic programming, we propose a semi-optimization algorithm by restricting the expansions. The tuple reduction for the sparse area are introduced to further eliminate the number of tuples. Through experiments, the proposed scheme can achieve twice throughput while only about quarter storage is required. Furthermore, the proposed scheme can fulfill OC-48 throughput with the 100K-filer database.

REFERENCES

[1] V. Srinivasan, G. Varghese and S. Suri, "Packet Classification using Tuple Space Search," in *ACM SIGCOMM*, September 1999, pp. 135–146.

[2] M. Waldvogel, G. Varghese, J. Turner and B. Plattner, "Scalable High Speed IP Routing Lookups," in *ACM SIGCOMM*, September 1997, pp. 25–36.

[3] Pankaj Gupta and Nick McKeown, "Packet Classification on Multiple Fields," in *ACM SIGCOMM*, September 1999, pp. 147–160.

[4] V. Srinivasan, G. Varghese, S. Suri and M. Waldvogel, "Fast Scalable Level Four Switching," in *ACM SIGCOMM*, September 1998, pp. 191–202.

[5] T.V. Lakshman and D. Stidialis, "High Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching," in *ACM SIGCOMM*, September 1998, pp. 203–214.

[6] Pankaj Gupta and Nick McKeown, "Packet Classification using Hierarchical Intelligent Cuttings," in *Hot Interconnects VII*, August 1999.

[7] Thomas Woo, "A Modular Approach to Packet Classification: Algorithms and Results," in *IEEE INFOCOM*, March 2000, pp. 1213–1222.

[8] V. Srinivasan and G. Varghese, "Fast IP lookups using controlled prefix expansion," *ACM Trans. On Computers*, vol. 17, pp. 1–40, Febuary 1999.

[9] M. Degermark, A. Brodnik, S.Carlsson, and S. Pink, "Small Forwarding Tables for Fast Routing Lookups," in *ACM SIGCOMM*, September 1997, pp. 3–14.

[10] P. Gupta, S. Lin, and N. McKeown, "Routing Lookups in Hardware at Memory Access Speeds," in *IEEE INFOCOM*, March 1999, pp. 1240–1247.

[11] S. Nilsson and G. Karlsson, "IP-Address Lookup Using $LC - Tries$," *IEEE JSAC*, vol. 17, no. 6, pp. 1083–1092, June 1999.

[12] NLANR Project, "National Laboratory for Applied Network Research.," See $http://www.nlanr.net$.