Performance Improvement of Two-Dimensional Packet Classification by Filter Rephrasing

Pi-Chung Wang, *Member, IEEE*, Chun-Liang Lee, Chia-Tai Chan, and Hung-Yi Chang

***Abstract*—Packet classification categorizes incoming packets into multiple forwarding classes in a router based on predefined filters. It is important in fulfilling the requirements of differentiated services. To achieve fast packet classification, a new approach, namely "filter rephrasing," is proposed to encode the original filters by exploiting the hierarchical property of the filters. Filter rephrasing could dramatically reduce the search and storage complexity incurred in packet classification. We incorporate a well-known scheme–rectangle search–with filter rephrasing to improve the lookup speed by at least a factor of 2 and decreases 70% of the storage expenses. As compared with other existing schemes, the proposed scheme exhibits a better balance between speed, storage, and computation complexity. Consequently, the scalable effect of filter rephrasing is suitable for backbone routers with a great number of filters.**

***Index Terms*—Firewalls, forwarding, packet classification, quality of service (QoS).**

I. Introduction

IN NEXT-GENERATON networks (NGNs), packet classification is important to fulfill the requirements of differentiated services. Using predefined filters, packet classification could categorize an incoming packet to a forwarding class (e.g., as indicated by the differentiated services codepoint (DSCP field) in the DiffServ model [1]). Secure filtering and service differentiation have been extensively employed to reflect the policies of network operations and resource allocation. The performance of packet classification is important in the deployment of NGNs; however, packet classification with a potentially large number of filters is complex and exhibits poor worst-case performance [2].

The problem of packet classification lies in multidimensional range match. To elaborate on the problem, the definition of the filter must be specified: a filter consists of a set of fields which, in turn, corresponds to another set of fields in the packet header. Each field could be a variable-length prefix bit string, a range, or an explicit value. A k-dimensional filter F is thus defined as $F = (f[1], f[2], \ldots, f[k])$. A filter might be any combination of fields; the most common fields are the source IP address (SA, 32 b), the destination IP address (DA, 32 b), the protocol type (8 b) and port numbers (16 b) of source/destination applications in the packet header. A packet P is said to match a particular filter F if for all i, the ith field of the header satisfies $f[i]$. Each filter has an associated action. For example, the filter $F = (140.113.*, *, \mathrm{UDP}, 1090, *)$ specifies a filter that addresses a flow that belongs to the subnet 140.113 and uses the progressive networks audio (PNA); the filter may assign the flow's packets with high queueing priority. Each action is usually assigned a cost to define its priority between the actions of the matching filters. The matching filter with the least-cost action will be enacted to process the arriving packets. Overall, the difficulty in packet classification is to search for this least-cost, matching filter.

A. Our Contributions

For solving the packet classification problem, we raise the importance of the scalability issue of which is pertinent to the potential needs of DiffServ, RSVP, load balancing, microflow recognition, multicast forwarding, network monitor, and subscriber management [3]–[6]. In this work, we present a scalable algorithm for 2-D (source prefix, destination prefix) packet classification. Our idea is inspired from the observation that the performance of the tuple-based schemes ties to the length distribution of the prefixes. Hence, we exploit the property of prefix nesting (or prefix containment [3]) by encoding the prefixes as well as the filters to revise the length distributions. In fact, the idea of encoding filters has been adopted in TCAM-based schemes to cope with the issue of filter replication for converting arbitrary range into prefixes [7], [8]. Contrarily, our idea applies to the algorithmic solution and simplifies the disordered structure of filters to improve speed and storage performance simultaneously.

The new hybrid scheme "filter rephrasing" can reorganize the entries in the hash tables to improve both search and storage complexity of the rectangle search. Although the effectiveness of the proposed scheme is based on the assumption that prefix nesting is rare, the convincing evidence based on real-world filter databases is also presented. In this work, we describe the idea of the filter rephrasing and combine it with a rectangle search [6], a well-known solution for packet classification. The new scheme is evaluated by both real-world and synthetic filter databases. In our experiments, it requires only 15% $\sim$ 30% storage of the rectangle search, while the lookup speed is improved by at least a factor of two with the encoding time included. We further compare the proposed scheme with the

Manuscript received April 26, 2004; revised May 24, 2005, and March 3, 2006.

P.-C. Wang is with the Department of Computer Science, National Chung Hsing University, Taichung 402, Taiwan, R.O.C. (e-mail: pcwang@cs.nchu.edu.tw).

C.-L. Lee is with the Department of Computer Science and Information Engineering, Chang Gung University, Taoyuan 333, Taiwan, R.O.C. (e-mail: cllee@mail.cgu.edu.tw).

C.-T. Chan is with the Institute of Biomedical Engineering, National Yang Ming University, Taipei 112, Taiwan, R.O.C. (e-mail: ctchan@ym.edu.tw).

H.-Y. Chang is with the Department of Information Management, National Kaohsiung First University of Science and Technology, Kaohsiung 811, Taiwan, R.O.C. (e-mail: leorean@ccms.nkfust.edu.tw).

Digital Object Identifier 10.1109/TNET.2007.893872

other existing schemes. The experimental results demonstrate that the proposed scheme exhibits a better balance between speed, storage, and computation complexity. Since the performance of the proposed scheme is irrelative to prefix length, the proposed scheme might be able to support IPv6 packet classification. Moreover, the scalable effect of the proposed scheme is suitable for applications, which requires numerous filters. A hybrid solution for supporting frequent filter updates is presented, and our source codes are publicly available.

The rest of this paper is organized as follows. Section II presents related work. Sections III and IV describe the tuple-based algorithms and the filter rephrasing, respectively. Section V presents the experimental setup and results. Finally, Section VI states our conclusion.

II. Related Work

Several algorithms for classifying packets have recently appeared in the literature [2], [6], [9]–[13]. They can be grouped into the following classes: hardware-based, trie-based, decision-tree, combination-based, and hash-based solutions. The following briefly describes the important properties of these algorithms. Assume that N is the number of the filters, k is the number of classified fields, and W is the length of the IP address.

A. Hardware-Based Solutions

A high degree of parallelism can be implemented in hardware to provide a speedup advantage. In particular, ternary content addressable memories (TCAMs) can be used effectively to look up filters. However, TCAMs with a particular word width cannot be used when flexible filter specification is required. Manufacturing TCAMs with sufficiently wide words to contain all bits in a filter is difficult. It also suffers from the problem of power consumption and scalability [4]. Lakshman and Stidialis presented another scheme that depends on a very wide memory bus [10]. The algorithm reads Nk bits from memory, corresponding to the best matching prefixes (BMPs) in each field, and determines their intersection to find a set of matching filters. The memory requirement for this scheme is $\mathrm{O}(N^2)$. The hardware-oriented schemes rely on heavy parallelism and involve considerable hardware cost. In addition, the flexibility and scalability of hardware solutions are very limited. In [3], Baboescu and Varghese addressed this issue and described an improved version by merging consecutive bits.

B. Trie-Based Solutions

Specifically for the case of two-field filters, Srinivasan *et al.* [9] presented a trie-based algorithm. The algorithm has a memory requirement of $\mathrm{O}(NW)$ and requires $2W-1$ memory accesses per lookup. Another version to support more than two dimensions is presented in [14]. In [15], the area-based quad tree (AQT) was also proposed for two-field filters. While the search performance of the native scheme is $\mathrm{O}(W \log N)$, it can be further improved to $\mathrm{O}(W + \log N)$ by using fractional cascading [15]. It is worth noting that AQT supports efficient $\mathrm{O}(\alpha \sqrt[\alpha]{N})$ update time, where α is a tunable integer parameter. The performance of the trie-based algorithms has been studied in [16].

C. Decision-Tree Solutions

Work on decision tree algorithms includes papers presented by Gupta and McKeown [13] and Woo [12]. Both schemes build a decision tree to categorize the filters into multiple sets. Since the number of filters in each set is limited by a predefined value, the linear search is applied to traverse the filters in a set. At each node, there is a rule for categorizing the filters, and the rule could specify a field [13] or a bit of any field [12]. In [17], the Hypercuts presented by Singh *et al.*further extends the single-dimensional rule in [13] into a multidimensional one. A suitable selection of rules would minimize the required storage and search time. However, due to the duplication of filters with wildcard fields, the space complexity is $\mathrm{O}(N^k)$.

D. Combination-Based Solutions

A general mechanism, called cross-producting, involves BMP lookups on individual fields and the use of a precomputed table to combine the results of individual prefix lookups [9]. However, this scheme suffers from a $\mathrm{O}(N^k)$ memory blowup for k-field filters. Gupta *et al.*presented an algorithm that can be considered to be a generalization of cross-producting [2]. After BMP lookup is performed, a recursive flow classification algorithm hierarchically performs cross-producting. Thus, k BMP lookups and $k-1$ additional memory accesses are required per filter lookup. The algorithm is expected to improve the average throughput significantly. Nevertheless, it requires $\mathrm{O}(N^k)$ memory in the worst case. In the case of two-field filters, this scheme is identical to cross-producting.

E. Hash-Based Solutions

The hash-based idea [6] has given rise to 2-D filters in a previous study [18]. The filters with specific prefix lengths are grouped into a tuple. Since each tuple has a specific bit length for each field, these bit lengths can be concatenated to create a hash key, which can be used to perform the tuple lookup. The matched filter can be found by probing each tuple alternately while tracking the least-cost filter. For example, the 2-D filters $F = (10*, 110*)$ and $G = (11*, 001*)$ both belong to the tuple $T_{2,3}$ in the third row and fourth column of the tuple space. When searching for $T_{2,3}$, a hash key is constructed by concatenating 2 b of the source field with 3 b of the destination field. Even a linear search in the tuple space represents a considerable improvement over a linear search of the filters since the number of tuples is typically much smaller than the number of filters. In the next section, the tuple-based algorithms are introduced in detail.

Among the existing algorithms, the ABV scheme [3] and Hypercuts [17] scheme are proposed to cope with the issues of performance degradation as the filters increase in number. The ABV scheme can provide sustainable throughput, but it requires relatively large storage and a wide memory bus to decrease the number of memory accesses. The Hypercuts scheme performs well under practical conditions; however, the computation complexity of the decision-based scheme is high even only a suboptimal cut calculation is adopted.

We can conclude two observations from the previous work.

- The existing solutions do not scale well in either speed, storage or computation complexity, or both. For example,

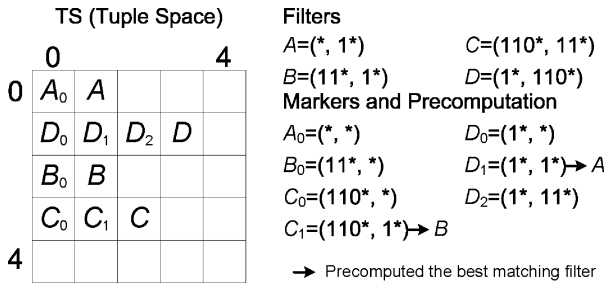

Fig. 1. Illustration of markers and precomputation.

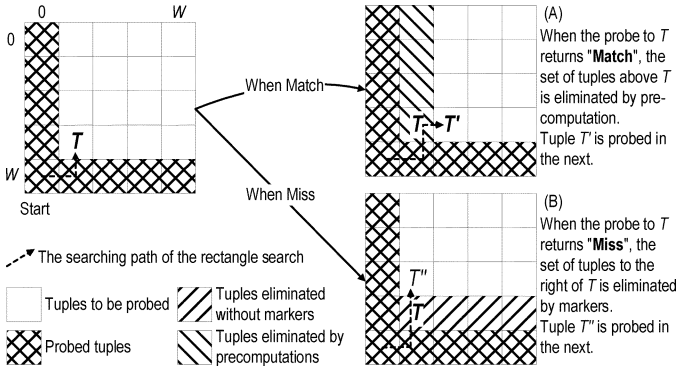

Fig. 2. Rectangle search algorithm.

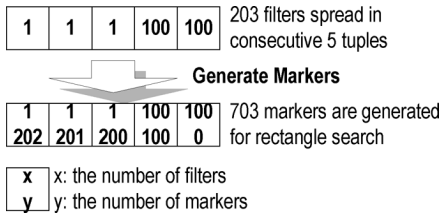

Fig. 3. Relationship between filters and markers.

the ABV scheme is not scalable in the required storage and the Hypercuts scheme requires exhausted computation.

- Most of the existing schemes rely on fast 1-D BMP lookups [2], [3], [9], [10], [13]. This is because the 1-D lookup is simpler than the search for two or more dimensions. Hence, it is reasonable to decompose the complex multidimensional search into multiple 1-D lookups.

III. Algorithms for Tuple Space Search

The simplest tuple-based algorithm, tuple pruning search, performs 1-D BMP lookups on individual fields to eliminate tuples that cannot match the query. To achieve this, the referred prefixes are collected with their locations in the tuple space to construct the pruning table for each field, where the lookup procedure starts. Then, the resulting locations of two dimensions are intersected. The tuples corresponding to the intersection will be probed. Since no extra entry is required except the pruning tables, it features low update cost. According to the authors' observation, the intersected tuples are very rare in the industrial filter database. Hence, the tuple pruning search performs well in the practical environment [6]. However, the worst-case performance $\mathrm{O}(W^2)$ remains the same as using a linear search.

A rectangle search was proposed to improve the performance of tuple pruning search [6]. Let filter $F = (f[1], f[2])$ with the length combination (i, j) belong to tuple $T_{i,j}$. A marker will be generated for each tuple left to $T_{i,j}$, say $T_{i,j'}, 0 \leq j' < j$, by eliminating the bits behind the j'th bits. After inserting the filters and markers, the entries at each tuple, say $T_{i,j}$, further precompute the least-cost matching filter among the tuples above $T_{i,j}$, say $T_{i',j}, 0 \leq i' < i$.

Fig. 1 shows an example of markers and precomputation using four filters A, B, C, and D. For each filter, a marker is inserted into each tuple left to the current tuple. Therefore, seven markers are generated for four filters. Consequently, the precomputation for each filter and marker is carried out. The marker C_1 thus precomputes the best matching filter, the one with lower cost between A and B, among the tuples above $T_{3,1}$ in the same column.

With markers and precomputation, the rectangle search can eliminate a set of tuples during each probing, as depicted in Fig. 2. Due to the precomputation of the best matching filter, tuples above T are eliminated if the probe of tuple T returns "match." Otherwise, tuples to the right of tuple T are ignored, since the absence of matching markers also indicates the nonexistence of matching filters. The time complexity of the rectangle search has been demonstrated to be $\mathrm{O}(2W-1))$, given a $W \times W$ tuple space, while the data structure would require a maximal number of NW markers.

Because of the complex precomputation, renewing the data structure of the rectangle search is inefficient. In Section IV-C, we present a hybrid approach to address this issue. The other issue of rectangle search is the number of generated markers caused by uneven filter length distribution. We show an extreme case with 203 filters, which occupy five tuples, as shown in Fig. 3. Each filter must generate one marker in each leftside tuple; hence, 703 markers are generated and result in explosive 906 entries. The size explosion is caused by an unbalanced distribution of filter lengths, which are the combinations of the prefix lengths. While the phenomenon has been identified in the 1-D routing prefixes [18]–[21], we believe that it is also held true for the 2-D filters.

To further clarify the assumption, the filter length distribution of an industrial filter database with 3028 filters illustrated in Fig. 4(a) can certify the unbalanced distribution. Our other databases also reveal the same findings. Fig. 4(b) shows the resulted distribution including the filters and markers. The widespread markers must be generated due to the need of guidance in the rectangle search and would cause storage explosion.

In [22], Wang *et al.*presented an algorithm to improve the required storage and the lookup speed of the rectangle search. Based on the observation that the performance of the rectangle search ties to the number of tuples, this scheme adopts a dynamic programming scheme to calculate the optimal set of tuples and reorganizes the rules by using rule expansion. However, the cost of precomputation would increase exponentially as the classifier expands and makes the scheme unsuitable for large rule sets.

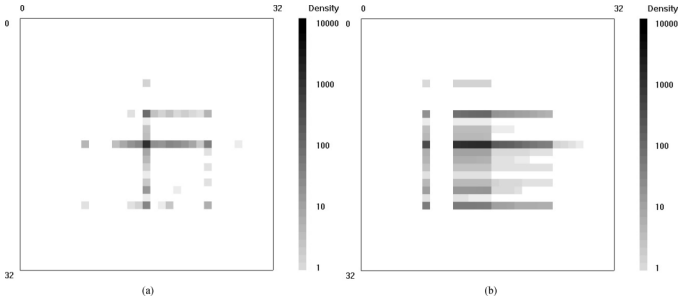

Fig. 4. Filter/marker length distribution for real filter database (3028 entries). (a) Length distribution of the original filters. (b) Length distribution of the original filters and the generated markers.

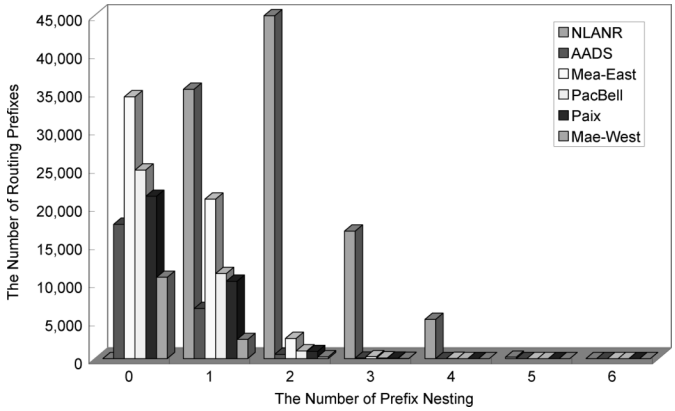

Fig. 5. Prefix nesting distribution of the routing tables.

In [23], the authors improved the rectangle search by exploiting the conflict-free constraint in the 2-D tuple space and introduced a binary search algorithm that exploits the conflict-free constraint in the 2-D tuple space. The conflict defines that a filter in a pair of filters is more specific than the other in one field and less specific in another field. To resolve the conflict, a conflict resolution filter is generated by concatenating the most specific fields in both fields [23]. For the filters in Fig. 1, filters A and B conflictin, which can be resolved by introducing a new filter $(110*, 110*)$.

The binary search scheme generates two different markers, one for the binary search within columns and the other for the binary search within the tuples in one column. Each search would traverse, at most, $\mathrm{O}(\log W)$ columns and $\mathrm{O}(\log W)$ tuples for each matching column; thus, the time complexity is $\mathrm{O}(\log^2 W)$. Although the binary search algorithm outperforms the rectangle search, the cost of table construction is high due to the exhaustive conflict detection [24], which is not preferable for large filter sets.

In summary, the tuple space search exploits an interesting property of the prefix length combination to improve packet classification; however, the existing tuple-based schemes are not without limitations. From what we have observed on the rectangle search, we are motivated to reorder the length distribution. We present a new scheme–filter rephrasing–to revise the length distribution by reducing the number of distinct prefix lengths as well as prefix length combinations. With filter rephrasing, both the speed and storage performance of the rectangle search can be dramatically improved.

IV. Tuple Reorganization by Filter Rephrasing

There are various approaches to reduce the number of distinct prefix lengths. For instance, a well-known prefix expansion capability based on duplicating entries is described in [19]. Dynamic programming is used to minimize the required storage resulting from duplication. The other algorithms [20], [21], [25] are based on a similar mechanism and have additional compression schemes built in. Briefly, these algorithms depend on either large storage or complex compression logic as a tradeoff for changing the length distribution.

The proposed "filter rephrasing" is inspired from the observation that the maximum number of prefix nesting is always much less than that of prefix lengths for each dimension. The nested prefix is defined as follows. Let $P_x = (a_1, a_2, \ldots, a_n, *)$ and $P_y = (b_1, b_2, \ldots, b_m, *)$ be two prefix strings, where $n \leq m$. Assume that the string matching function $\text{P-match}(P_x, P_y)$ is used to find the substring of P_y. P_x is a nested prefix of P_y if $\text{P-match}(P_x, P_y) = P_x$. In [3], Baboescu and Varghese surveyed that no prefix contains more than four matching nested prefixes in each dimension of the ISP/firewall filter databases. The characteristic conforms to the nature of the network-layer address/mask specification described in [26]. This is mainly because the service boundaries specified by the filters highly relate to those specified by the network-layer prefixes, for example, the VPN filters in ISP databases. The major exceptions are the filters for specifying designated-server or interdomain traffic, where the former requires full-length network-layer addresses and the latter use numerous wildcards. Such filters occupy a great portion in intranet firewalls [14], [27], though the property of few nested prefixes is not affected.

We also illustrate the number of nested prefixes for each routing prefix in the existing routing tables from some major network access points in Fig. 5. For most routing prefixes, there are usually less than three nested prefixes in the routing tables and six in the worst case. Our observation for industrial filter databases also presents the same results and is consistent with the existing literature [3], [11], [28]. In the previous work, the property of prefix nesting has been applied to improve the search performance of packet classification [3], [9], [14]. In this work, we further exploit this hierarchical property to improve the lookup speed and required storage simultaneously.

Before illustrating the principle of design, the nature of IP routing prefixes should be clearly stated. The adoption of classless interdomain routing (CIDR) [29] allows the network administrator to specify a subnet within an existing network. For example, an ISP network is specified by prefix $\langle 206.95.* \rangle$, whose next hop is A. An enterprise network, which is specified by another prefix $\langle 206.95.130.* \rangle$, might exist and its next hop is B. The new scheme must be able to reflect the hierarchical nature of the routing prefixes. Likewise, the generated key for $\langle 206.95.* \rangle$ must be a prefix of that for $\langle 206.95.130.* \rangle$.

A straightforward scheme is to divide the prefixes into several bitstreams according to the lengths of their nested prefixes. An example is used to explain the procedure, as shown in Fig. 6. The prefix P_4 $\langle 010000 \rangle$ has two nested prefixes P_1 $\langle 0 \rangle$ and P_3 $\langle 0100 \rangle$. Thus, P_4 is divided as three bitstreams $\langle 0 \rangle$, $\langle 100 \rangle$,

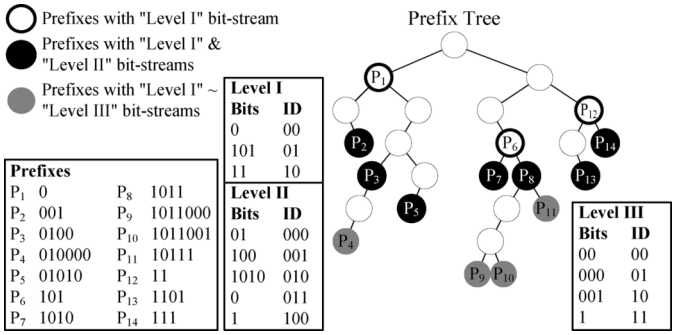

Fig. 6. Basic prefix encoding.

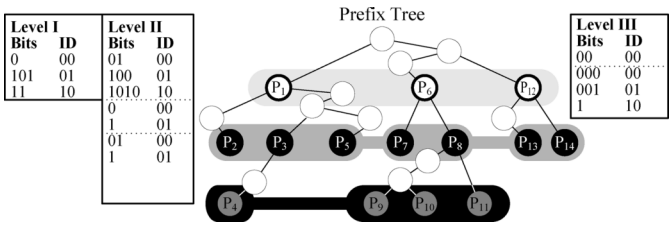

Fig. 7. Exclusive prefix encoding.

and $\langle 00 \rangle$ that are inserted to the bit-stream groups "Level I," "Level II," and "Level III," respectively. Clearly, it may derive the same bitstreams from different prefixes; such as the "Level II" bit-stream $\langle 01 \rangle$ of prefix P_{13} is identical to that of P_2. In each group, the duplicate bitstreams must be eliminated, then each bitstream is assigned to a new identification (ID). By concatenating the relevant IDs, the encoded key of each prefix is generated. In this example, there are three different bitstreams in "Level I," five in "Level II," and four in "Level III." Therefore, maximal 7 $(= 2 + 3 + 2)$ bits are required to represent the original prefixes, but the number of distinct lengths is reduced from 7 to 3.

The basic scheme is simple, but it might overestimate the number of the encoded bitstreams without considering the associated relations between prefixes. For example, the bitstreams $\langle 100 \rangle$ and $\langle 1010 \rangle$ (i.e., corresponding to P_3 and P_5, respectively) in "Level II" only concatenate to $\langle 0 \rangle$ (i.e., P_1). Thus, we only have to count for the number of bitstreams attached to a specific "shorter" bitstream. In our example, there are three bitstreams connected to $\langle 0 \rangle$ (P_1) and two to both $\langle 101 \rangle$ (P_6) and $\langle 11 \rangle$ (P_{12}). Accordingly, the number of bits for "Level II" IDs could be reduced to 2 and the maximal length is further reduced to 6. In Fig. 7, we list the IDs for all bitstreams. The dotted lines are used to separate the bitstreams based on their concatenated prefixes at each level.

In Table I, we compare the lengths of the encoded prefixes in the original, basic, and exclusive schemes. Note that efficient encoding would not affect the number of distinct lengths, but could reduce the storage requirements for indices in the tuple space.

The original prefixes of the filters are replaced by the encoded ones, and the new filters are used to construct the tuple space. The length distribution of the new filters and their markers from Fig. 4(b) is shown in Fig. 8. For the prefixes used in the industrial filter database, the maximal number of prefix nesting is 2.

TABLE I
ENCODED PREFIXES FOR DIFFERENT SCHEMES

Prefix	Original	Basic	Exclusive
P_1	0	00	00
P_2	001	00000	0000
P_3	0100	00001	0001
P_4	010000	0000100	000100
P_5	01010	00010	0010
P_6	101	01	01
P_7	1010	01011	0100
P_8	1011	01100	0101
P_9	1011000	0110001	010100
P_{10}	1011001	0110010	010101
P_{11}	10111	0110011	010110
P_{12}	11	10	10
P_{13}	1101	10000	1000
P_{14}	111	10100	1001

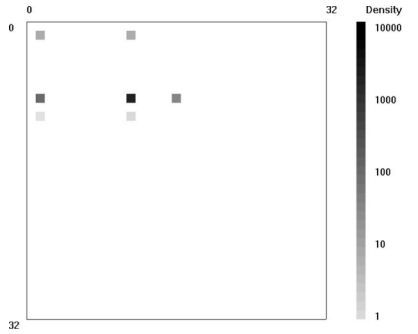

Fig. 8. Length distribution of the encoded filters and markers.

Hence, the number of distinct lengths in both dimensions is 3. The encoded filters only occupy seven tuples, rather than 46 with the original filters. The markers are inserted into only four tuples which, in turn, reduce the required storage dramatically. Also, performing a rectangle search in the encoded filters is enhanced since the number of rows and columns are lessened.

A. Filter Rephrasing Algorithm

The procedure of filter rephrasing consists of two parts: "ID assignment" and "prefix encoding." A simple "ID assignment" procedure starts from dividing the referred prefixes into multiple bitstreams according to their nested prefixes. Then, the number of distinct "Level I" bitstreams N_1 is calculated and a new ID bitstream whose length is $\log_2(N_1)$ is assigned. Next, the bitstreams in "Level II" are considered. Unlike the "Level I" bitstreams, the "Level II" bitstreams are grouped according to their directly engaged bitstreams. The number of the bitstreams in the largest group is set as N_2, and a new ID with length $\log_2(N_2)$ is assigned to each "Level II" bitstream. Note that the bitstreams in different groups might be assigned to the same ID, but they will not cause ambiguous encoding since their engaged bitstreams are distinct. The procedure is repeated until all bitstreams are assigned to new IDs. The "prefix encoding" then rephrases the prefixes by replacing their bitstreams with new IDs. After that, the new filters are generated by renewing the prefixes with rephrased ones, and the precomputation of the rectangle search constructs the searchable data structure based on the new filters. We present the detailed algorithm based on the binary tree in Fig. 9. Since each node is traversed exactly

```
ID Assignment Algorithm
Input: The root of the binary tree constructed from the prefixes specified
       in the filters
Output: The IDs for the prefixes and the required bits for each level
Assign (Root,Count,Level) BEGIN
IF (Root→Prefix = true) BEGIN
        Root→ID=Count+1;
        M=Assign(Root→Left,0,Level+1);
        N=Assign(Root→Right,M,Level+1);
        If (N>Max_Level[Level+1])
                Max_Level[Level+1]=N;
        return (Count+1);
END
ELSE BEGIN
        M=Assign(Root→Left,Count,Level);
        N=Assign(Root→Right,M,Level);
        return N;
END
/* The number of the required bits at level i equals to log2(Max_Level[i]).
The encoded prefixes are derived by concatenating IDs, which are fetched
along the path from the root to the prefix node in the binary tree. */
```

Fig. 9. ID assignment algorithm.

once, the time complexity for ID assignment is $O(NW)$. In addition, each prefix encoding is performed by traversing the binary tree from root to the node of the encoded prefix and concatenating the IDs along the path. Hence, the time complexity is also $O(NW)$.

B. Search Procedure and Implementation

To search for the encoded filters, the source/destination addresses of the incoming packet must be encoded by performing 1-D BMP lookups upon the searchable data structure of the original prefixes. The result of each lookup is the encoded prefix. The rectangle search is then performed upon the encoded prefixes. Although the extra lookups are required, its cost is low. As summarized in Section II, numerous schemes use fast 1-D lookups to conduct the multidimensional search in packet classification [2], [3], [9], [10], [13]. Furthermore, the required entries for 1-D lookups are much fewer than filters because each prefix is usually used by various filters [11]. The fast 1-D lookup algorithm proposed in the previous schemes can be applied to provide good performance, such as [18], [19], and [30]. For IPv6 scalability, we adopt the binary search on hash tables, which can accomplish each lookup within five hash accesses for IPv4 and seven for IPv6 [18].

The proposed scheme can be implemented with software or hardware. With software implementation, the number of memory accesses is defined as

$$\begin{aligned}\text{Memory Accesses} = {} & \text{BMP(SA)} \cdot \text{access} \\ & + \text{BMP(DA)} \cdot \text{access} \\ & + \text{RS(encoded filter)} \cdot \text{access} \end{aligned} \quad (1)$$

where $\text{BMP}() \cdot \text{access}$ and $\text{RS}() \cdot \text{access}$ are the number of memory accesses for the BMP lookup and rectangle search, respectively. Typically, each BMP lookup and rectangle search need $\log W$ [18] and $2S - 1$ hash accesses [6], where S is the maximal number of prefix nesting. The throughput of the packet classification can be further improved by pipelining all hash accesses. With a proper selection of hash function [31], it is possible to allow a speed of one lookup per memory access at the cost of possibly increased complexity.

Note that the generated entries in the rectangle search only ties to the number of occupied columns in the tuple space. Therefore, a heuristic can select the field with more distinct prefix lengths as the horizontal axis in tuple space and perform prefix encoding only on this field. Since only one field is encoded, only one BMP lookup is required. The number of generated markers is the same as our original design; nevertheless, the number of tuples would be increased and the speed improvement is lessened. Another heuristic is to choose the field of destination prefix as the horizontal axis of the tuple space and combine the BMP lookup with IP address lookup, which is performed for each incoming packet, to minimize the overhead of the BMP lookup.

C. Handling Dynamic Filters

To address the problem of filter updates, we classify the filter updates into three categories: priority/action change, insertion, and deletion. Since each priority/action change can be carried out by filter deletion and filter insertion, we only describe the procedures for the latter two updates.

We begin with the procedure of filter insertion. There are two reasons to make the insertion of a filter into the data structure of the proposed scheme difficult. The first is the complex precomputation for both new and existing markers. For each inserted filter, at most W new markers are generated and each marker must precompute the best-matching filter on the same column of the tuple space, as depicted in Fig. 1. In addition, the inserted filter might cover all existing filters in the worst case; thus, the best-matching filter carried in at most N existing markers on the same column must be revised. Second, the prefixes in the inserted filter may not exist in the original filter database; thus, a new ID must be assigned to the new prefix. In the worst case, the new prefixes would change the prefix nesting and result in re-encoding all prefixes as well as reconstructing the data structure.

Since inserting a filter into the tuple pruning search is simple [6], we propose a hybrid approach by maintaining two tuple spaces—one for the encoded filters and the other for the original filters. The tuple space of the encoded filters is searched by rectangle search and the other is searched by the tuple pruning search.

Initially, the tuple pruning search stores all filters as the rectangle search does; however, the new filter is only inserted into the data structure of the tuple pruning search without affecting that of the rectangle search. Each prefix in the pruning table maintains two location lists, one for all filters and one for the new filters. The location of the new filter in the tuple space is inserted into both location lists. For each packet classification, the rectangle search and tuple pruning search are performed simultaneously, whereas the tuple pruning search only considers the location lists of the new filters. Either of the results with a lower cost is activated, as depicted in Fig. 10. Once the lookup performance of tuple pruning search degrades to a predefined threshold, the new filters are merged into a rectangle search and trigger reconstruction. During the period of reconstruction, the

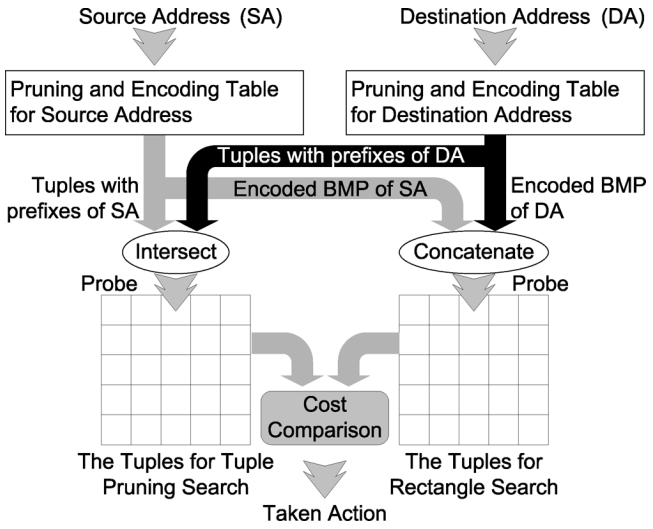

Fig. 10. Combining the rectangle search with the tuple pruning search to support filter updates.

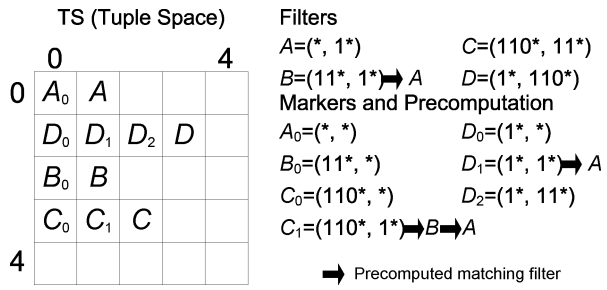

Fig. 11. Example of filter deletion.

tuple pruning search looks up all filters by referring to the location lists of all filters to achieve continuous packet forwarding.

During reconstruction for the rectangle search, the location lists of the new filters in tuple pruning search should be emptied out. However, we can avoid this procedure by recording the timestamp of reconstruction in each location list of new filters. Once the location list of the new filters is fetched for either searching or inserting filters, its timestamp is checked and the list, whose timestamp is older than the timestamp of the most recent reconstruction, will be dropped.

In the following, we present the procedure of filter deletion. To support deletion, the precomputation of the least-cost matching filter for each filter and marker is modified to "precompute and record all matching filters." The matching filters are listed according to their cost in ascending order. Also, the deleted filter with its precomputed best matching filter is inserted into the tuple pruning search as a new filter. For an incoming address pair, which is matched to the deleted filter, the tuple pruning search will retrieve the deleted filter and its best matching filter, which also matches the incoming address pair. Then, the rectangle search would notify that a possible match of deleted filter exists. Consequently, whenever the deleted filter is found as the first precomputed matching filter in any traversed marker and filter, it is removed and the its following precomputed filter is fetched. However, if the deleted filter is not the first precomputed filter, it would be reserved in the rectangle search.

Consider the example in Fig. 11. The precomputation for filter A and marker C_1 is modified by adding filter A for possible matching. Once filter B is deleted, filter B and its best matching filter A are inserted into the tuple pruning search. For an address pair $(1101, 1100)$, both filters B and A are retrieved in the tuple pruning search, and filter B is recognized as deleted. In the rectangle search, the marker C_1 will be retrieved, and its best matching filter is the deleted filter B. Therefore, the filter B is removed from the list of C_1 to retrieve the next matching filter A. As a result, the rectangle search can guarantee that the matching filter always exists.

In [16], the authors have reported that the tuple pruning search could handle each filter insertion within 100 μs. Therefore, our hybrid solution could handle 100-K filter insertion/deletion per second. Also, our source code can reconstruct the data structure of the proposed scheme for 100-K filters within 2 s by a generic platform equipped with a 2-GHz CPU. Therefore, it is possible for the proposed scheme to handle frequent updates without significantly decreasing the performance.

V. Performance Evaluation

In this section, we compare the proposed scheme with the existing schemes. The performance metrics include the required storage—the numbers of memory accesses in average and in the worst case. Since the proposed scheme must perform BMP lookups, the memory accesses and required storage of the 1-D searches are counted. In the first part, the proposed scheme is compared with the rectangle search to show the effectiveness of filter encoding. In the second part, we further demonstrate the performance of the proposed scheme by comparing it with other existing schemes. The generated filter databases and source codes are published in the following web site: http://www.csie.nctu.edu.tw/~leecl/pclass.

A. Comparison With Rectangle Search

What filter encoding can achieve in performance is best illustrated through the comparison with the rectangle search. We use the industrial and synthetic filter databases to investigate their performance. The industrial filter databases are downloaded from the Abilene Observatory Project [32], which supports the collection and dissemination of network data associated with the Abilene Network [33]. The observatory serves network engineers by providing a view of the operational data associated with a large-scale network. Ten databases are collected, and the number of filters varies from 331 to 3028. The synthetic filter databases are adopted to stress the scalability of the proposed scheme. We repeat how Baboescu and Varghese generated their databases in [3]. Twelve (source prefix, destination prefix) synthetic filter databases are generated by randomly sampling the routing prefixes in a routing table downloaded from NLANR [34]. The minimal filter database contains 1-K filters, and the sizes of the databases increase to 5 K, 10 K, 20 K, 30 K, and so on, and up to 100-K filters.

The required tuples and entries based on the industrial filter databases are listed in Table II. The fields, hash table, and hash entry present the required storage for 1-D BMP lookups. The occupied tuples of the proposed scheme are reduced dramatically

TABLE II
COMPARING STORAGE

Filter Databases	Rectangle Search				Proposed Scheme					
	Tuple	Entry	Row	Col	Tuple	Entry	Row	Col	Hash Table	Hash Entry
331	104	1,862	8	15	5	383	2	3	23	358
548	154	3,386	12	15	9	638	4	3	29	583
651	170	5,274	12	17	7	759	3	3	30	590
722	175	4,226	14	17	7	839	3	3	30	571
986	144	4,924	11	16	4	993	3	2	28	1,000
1,130	52	5,250	5	17	3	1,137	2	2	21	1,084
1,142	218	8,225	14	19	9	1,308	4	3	33	947
2,278	221	12,436	16	17	8	2,514	3	3	32	1,486
2,300	230	12,551	16	18	9	2,534	4	3	33	1,604
3,028	210	16,166	16	20	7	3,276	3	3	35	1,791

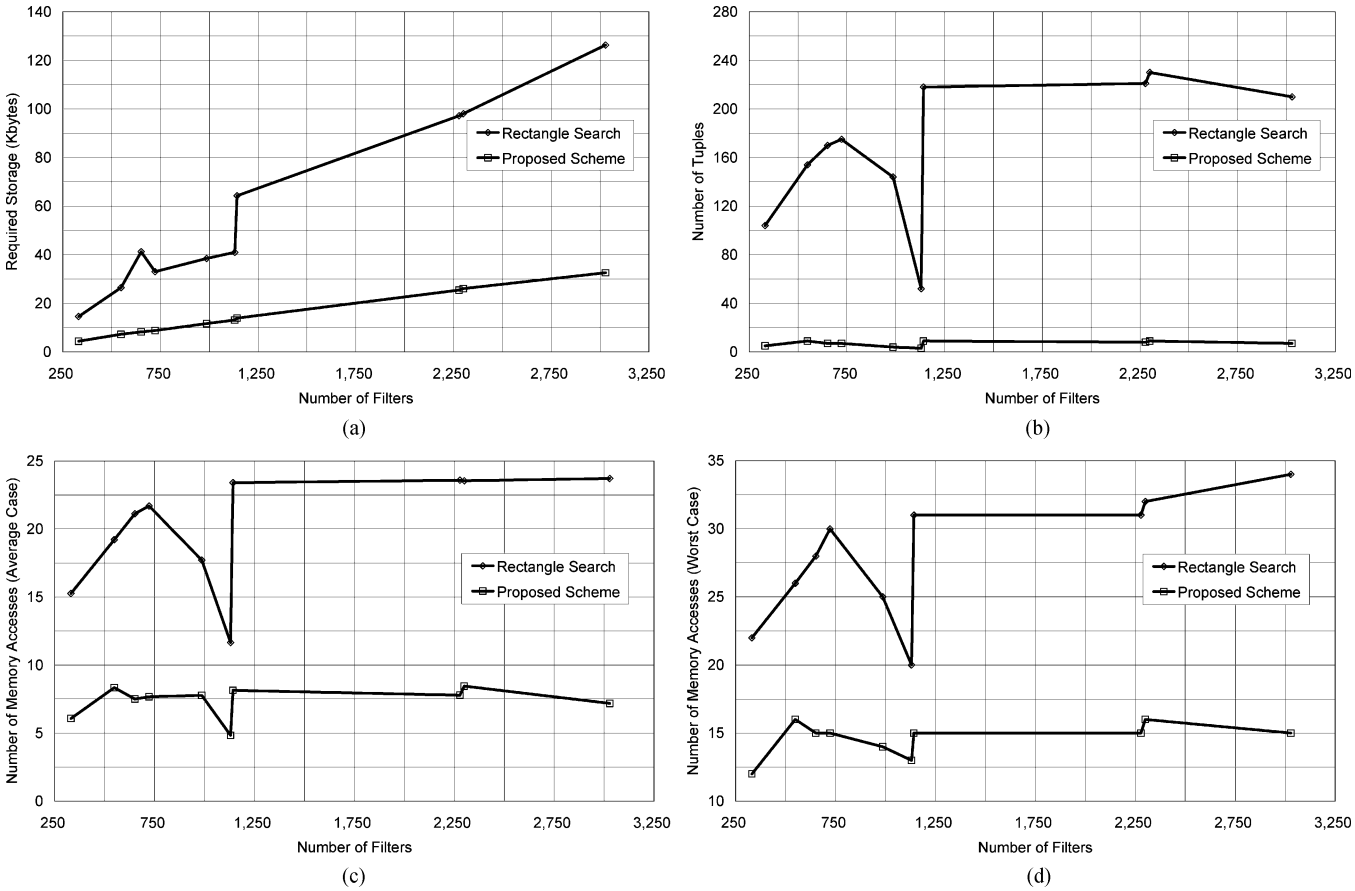

Fig. 12. Performance comparisons for the industrial filter databases. (a) Required storage. (b) Number of tuples. (c) Average performance. (d) Worst performance.

due to the elimination of rows and columns by filter rephrasing. Also, the rectangle search requires markers as many as five to seven times that of filters, whereas the proposed scheme only needs a maximum 17% of the filters. Even when the storage for BMP lookups is counted, the proposed scheme still outperforms the rectangle search significantly.

The required storage and tuples are also illustrated in Fig. 12(a) and (b). In the rectangle search, the number of required entries is increased along with the number of filters. The curve of the rectangle search fluctuates due to the characteristics of different filter databases. The number of filters and markers of the proposed scheme is smaller than that of the rectangle search, mainly because filter rephrasing eliminates the distinct lengths of filters and leads to fewer entries. Also, the increasing rate for the proposed scheme is moderate; it reflects the fact that the number of prefix nesting in each dimension is less than that of distinct lengths. The findings demonstrate that filter rephrasing supports large filter databases since the number of generated markers is proportional to the number of filters instead of the distinct lengths.

Fig. 12(c) and (d) shows the search performance in average and in the worst case. For both schemes, Fig. 12(b) depicts that the tuple distributions highly relate to the lookup performance. However, the performance of the proposed scheme is relatively stable. The enhancement comes from the stable distribution of the nested prefixes. Through encoding filters, the native rectangle search is reformed to a near-optimal level.

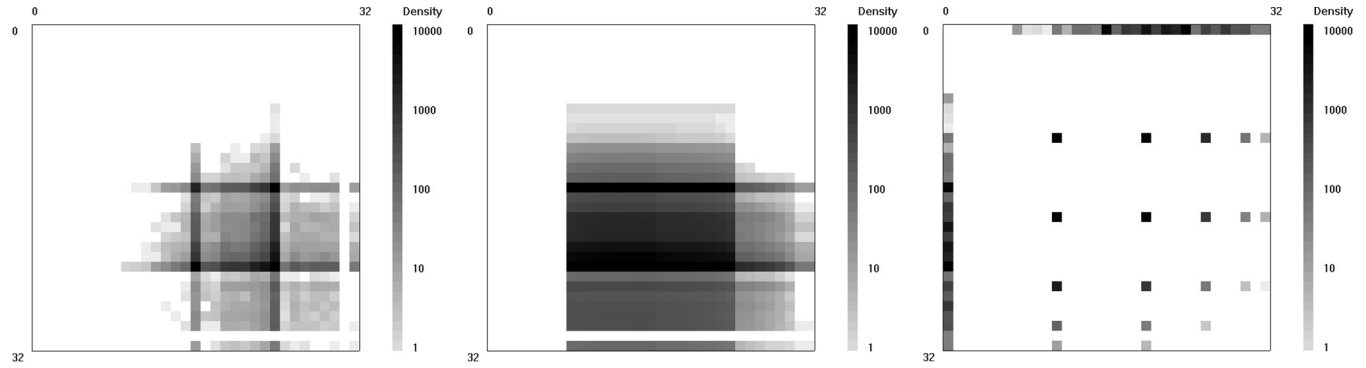

Fig. 13. Filter length distribution of synthetic filter databases (100-K entries). (a) Original filters. (b) Filters and markers. (c) Encoded filters, markers, and 1-D hash entries.

TABLE III
COMPARING STORAGE

Filter Databases	Rectangle Search				Proposed Scheme					
	Tuple	Entry	Row	Col	Tuple	Entry	Row	Col	Hash Table	Hash Entry
1K	304	7,558	20	19	9	1,431	3	3	44	1,430
5K	379	47,163	20	21	9	6,719	3	3	48	6,699
10K	482	115,660	23	23	13	13,932	4	4	50	12,641
20K	523	264,106	24	24	21	31,134	5	5	50	24,048
30K	535	404,771	24	25	19	46,683	4	5	50	34,612
40K	528	538,987	24	24	21	61,139	5	5	50	44,603
50K	545	678,286	24	24	20	76,817	5	5	50	54,182
60K	534	838,153	24	24	21	88,238	5	5	50	63,419
70K	548	995,468	24	24	22	100,386	5	5	50	71,756
80K	552	1,151,962	25	24	21	114,911	6	5	50	80,314
90K	539	1,311,341	24	24	21	126,810	5	5	50	89,210
100K	563	1,467,955	25	25	21	140,338	6	5	50	97,293

Next, we evaluate performance with the synthetic filter databases. Fig. 13 shows the tuple distribution for the 100-K-entry filter databases. Most filters are in the region from the upper left tuple (16,16) to the bottom right tuple (24,24) because most routing prefixes have a length of 16–24 b [18], as shown in Fig. 13(a). In Fig. 13(c), the entry distribution of the 1-D hash entries is illustrated in the bricks at the leftmost and upmost margins. Also, due to the volume of prefixes, the length of the encoded prefixes is longer than 32 b. Hence, we scale the length of the encoded filters into 32 b to simplify the representation. While the rectangle search enlarges the number of occupied tuples due to the large amount of markers in Fig. 13(b), the proposed scheme avoids such unnecessary tuples since the length combinations of the encoded filters are more regular. Also, memory management can be simplified with fewer tuples.

We present the required storage and entries in Table III. The proposed scheme effectively reduces the occupied tuples and generated markers by eliminating the number of rows and columns. For the synthetic databases, the proposed scheme generates the number of markers that are only 40% of the filters, while the rectangle search requires markers that are 14 times the filters. The number of the required markers in the rectangle search is greatly increased as well as the number of distinct lengths. Nevertheless, the proposed scheme is only affected by the number of prefix nesting and requires only less than one-tenth markers as in the rectangle search.

In Fig. 14, the comprehensive results are presented. Fig. 14(a) and (b) presents the required storage and tuples. Since the lengths of the random generated filters are uniformly distributed, all of the curves are smooth. For the native rectangle search, the number of required entries is proportional to both the number of filters and the number of distinct prefix lengths. The proposed scheme removes the affection of prefix lengths by the prefix nesting counts and features low storage requirements. The search performance in average and the worst cases are illustrated in Fig. 14(c) and (d), respectively. As the size of databases reaches a certain value, the memory accesses of the rectangle search and the proposed scheme would enter a steady state, which can be no longer influenced by the number of filters. This is also a well-known advantage of the tuple-based algorithms. The proposed scheme benefits both from the superiority of the rectangle search and further achieves better performance in both average and worst cases.

For IPv6 packet classification, the search speed of the rectangle search could be considerably degraded with an increased number of distinct prefix lengths, which only affects the performance of 1-D lookups for the proposed scheme. Therefore, we presume that the proposed scheme is a compelling solution for IPv6 packet classification by making a reasonable assumption that the prefix nesting property will continue to hold with IPv6 [21].

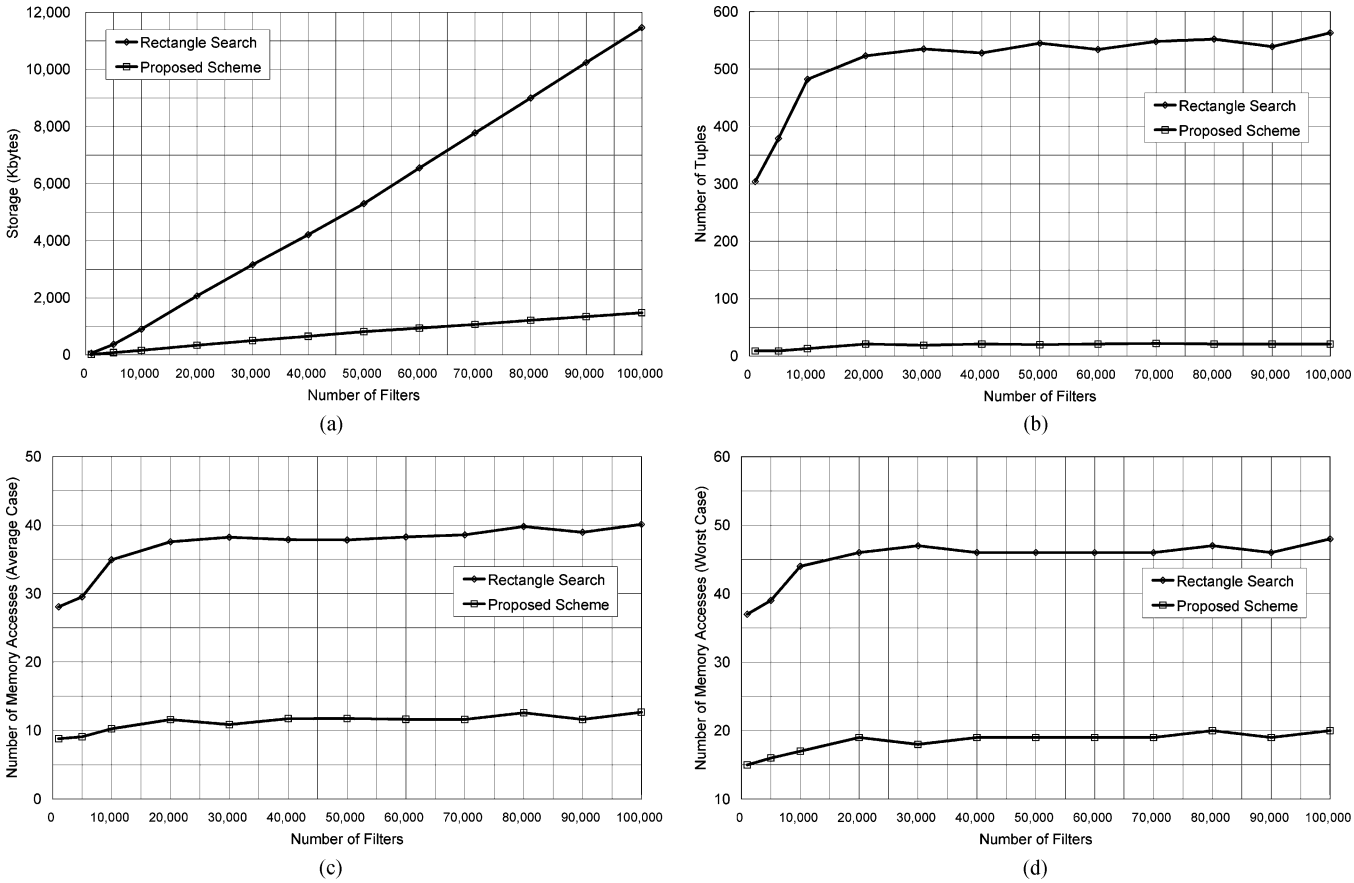

Fig. 14. Performance comparisons for industrial filter databases. (a) Required storage. (b) Number of tuples. (c) Average performance. (d) Worst performance.

B. Comparison of Other Existing Schemes

In the following experiments, we evaluate the performance against the notable existing schemes, including Grid of Tries [9], ABV [3], and HiCuts [13]. The source code of Grid of Tries is available in our website, and the codes of ABV and HiCuts are available in authors' web sites [35], [36]. For 2-D filter databases, the methodology between HiCuts and HyperCuts is quite similar [17]; thus, the experimental results of HiCuts could also reflect the performance of HyperCuts. The tested filter databases are synthesized by a public tool, ClassBench [28]. The tool includes 12 seed files derived from real filter sets to reflect the characteristics of filters for different applications, including the access control list, firewall, and IP chain. For each parameter file, we generate a filter set with 50-K distinct 2-D filters. In addition, the filter sets are randomly rearranged to represent their priority.

As shown in Table IV, the proposed scheme usually features the least storage. The performance of Grid of Tries ties to the maximal prefix length of each field. Since the acl and ipc seeds frequently generate 32-b prefixes in the filters, the search performance and required storage also degrade for these filters' databases. The ABV scheme requires the most storage for databases generated by acl and ipc seeds. This is because the acl and ipc seeds generate the most distinct prefixes in both fields and cause a significant increase in the numbers of bit vectors. The average search performance can be improved by implementing filter rearrangement [3], which is not provided in the source code. However, the required storage of ABV makes it unsuitable for some applications, which might specify more distinct prefixes. The performance of HiCuts fluctuates for different types of seeds. For example, HiCuts results in minimal storage for acl1 and acl5 databases, but it also suffers from large storage for firewall databases. Therefore, the HiCuts scheme may require more efforts on adjusting parameters (e.g., space factor) (spfac) and bucket size (binth) [13]. Compared to existing schemes, the proposed scheme could achieve good performance in both speed and storage without complex computation.

VI. Conclusion

Packet classification is a highly effective primitive process that is associated with a policy-defined context to activate differentiated services. This study investigates the properties of the filters and proposes "filter rephrasing" to improve performance and storage of the rectangle search algorithm. The idea is to reduce the length combinations of the filters by encoding their referred prefixes. By regulating the lengths of the prefixes, the required tuples and entries for the rectangle search can be reduced significantly. Despite the requirement of two 1-D BMP lookups, we have shown that the cost is low with respect to the advantages gained in performance enhancement. A solution for supporting frequent filter updates by combining the tuple pruning

TABLE IV
PERFORMANCE COMPARISON WITH THE EXISTING SCHEMES

	Grid of Tries			ABV (word size=128 bits)			HiCuts (spfac=1,binth=64)			Proposed Scheme		
	Speed		Storage	Speed		Storage	Speed		Storage	Speed		Storage
seed	Max	Avg	Mbytes	Max	Avg	Mbytes	Max	Avg	Mbytes	Max	Avg	Mbytes
acl1	65	61.82	4.5	29	12.87	390.6	40	12.44	0.9	17	15.19	2.6
acl2	65	49.27	3.9	94	33.41	300.5	35	11.01	3.4	22	17.58	2.0
acl3	65	55.24	7.2	156	62.30	479.8	55	16.40	9.7	20	15.66	2.7
acl4	65	55.38	3.8	111	41.75	316.2	20	7.29	5.9	24	20.96	2.1
acl5	65	62.03	5.9	26	12.06	332.9	68	22.12	0.7	17	15.00	1.2
fw1	65	45.89	4.3	399	245.52	103.5	28	10.20	78.8	18	14.37	1.7
fw2	65	43.15	3.8	293	163.62	109.5	11	6.13	34.2	16	12.90	1.5
fw3	59	40.49	3.4	370	219.63	73.8	35	8.74	97.7	16	14.27	1.4
fw4	65	42.30	4.1	401	166.85	139.3	28	11.30	49.2	17	14.33	1.7
fw5	65	45.54	3.6	364	216.45	69.0	34	10.89	109.5	19	16.70	1.5
ipc1	65	53.40	7.1	104	44.33	504.2	26	10.21	7.7	19	14.45	2.8
ipc2	65	49.95	5.4	457	34.43	198.9	69	27.44	13.1	14	12.90	1.8

search and rectangle search is presented. The experimental results obtained for industrial and synthetic databases show that the lookup speed is increased by a factor of two while the required storage is reduced to only about one-fifth. Comparing the existing schemes further demonstrates the effectiveness of the proposed scheme. Since the performance of the proposed scheme is almost irrelative to prefix length, it might be a feasible solution for IPv6 packet classification. For all of these reasons, the proposed scheme based on filter encoding seems to be a scalable solution for the large filter databases of the next-generation applications. The idea of encoding filters might be applied to other algorithmic schemes. Yet, the tradeoff between the performance improvements and the extra 1-D lookups need to be further justified in the future.

REFERENCES

[1] S. Blake *et al.*, "An architecture for differentiated services," RFC 2475, 1998.

[2] P. Gupta and N. McKeown, "Packet classification on multiple fields," in *Proc. ACM SIGCOMM*, Sep. 1999, pp. 147–160.

[3] F. Baboescu and G. Varghese, "Scalable packet classification," in *Proc. ACM SIGCOMM*, Aug. 2001, pp. 199–210.

[4] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Netw. Mag.*, vol. 15, no. 2, pp. 24–32, Mar./Apr. 2001.

[5] V. P. Kumar, T. V. Lakshman, and D. Stiliadis, "Beyond best effort: Router architectures for the differentiated services of tomorrow's internet," *IEEE Commun. Mag.*, vol. 36, no. 5, pp. 152–164, May 1998.

[6] V. Srinivasan, G. Varghese, and S. Suri, "Packet classification using tuple space search," in *Proc. ACM SIGCOMM*, Sep. 1999, pp. 135–146.

[7] J. van Lunteren and T. Engbersen, "Fast and scalable packet classification," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 4, pp. 560–571, May 2003.

[8] H. Liu, "Efficient mapping of range classifier into ternary-CAM," in *Proc. Hot Interconnects X*, Aug. 2002, pp. 95–100.

[9] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast scalable level four switching," in *Proc. ACM SIGCOMM*, Sep. 1998, pp. 191–202.

[10] T. V. Lakshman and D. Stidialis, "High speed policy-based packet forwarding using efficient multi-dimensional range matching," in *Proc. ACM SIGCOMM*, Sep. 1998, pp. 203–214.

[11] A. Feldmann and S. Muthukrishnan, "Tradeoffs for packet classification," in *Proc. IEEE INFOCOM*, Mar. 2000, pp. 1193–1202.

[12] T. Woo, "A modular approach to packet classification: Algorithms and results," in *Proc. IEEE INFOCOM*, Mar. 2000, pp. 1213–1222.

[13] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," presented at the Hot Interconnects VII, Aug. 1999.

[14] F. Baboescu, S. Singh, and G. Varghese, "Packet classification for core routers: Is there an alternative to CAMS?," in *Proc. IEEE INFOCOM*, Mar. 2003, pp. 53–63.

[15] M. Buddhikot, S. Suri, and M. Waldvogel, "Space decomposition techniques for fast layer-4 switching," in *Proc. IFIP 6th Int. Workshop on Protocols for High Speed Networks*, Aug. 1999, pp. 25–41.

[16] V. Sahasranaman and M. Buddhikot, "Comparative evaluation of software implementations of layer-4 packet classification schemes," in *Proc. IEEE ICNP*, Nov. 2001, pp. 220–228.

[17] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *Proc. ACM SIGCOMM*, Aug. 2003, pp. 213–224.

[18] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing lookups," in *Proc. ACM SIGCOMM*, Sep. 1997, pp. 25–36.

[19] V. Srinivasan and G. Varghese, "Fast IP lookups using controlled prefix expansion," *ACM Trans. Comput.*, vol. 17, pp. 1–40, Feb. 1999.

[20] P. Gupta, S. Lin, and N. McKeown, "Routing lookups in hardware at memory access speeds," in *Proc. IEEE INFOCOM*, Mar. 1999, pp. 1240–1247.

[21] S. Nilsson and G. Karlsson, "IP-address lookup using *LC – Tries*," *IEEE J. Sel. Areas Commun.*, vol. 17, no. 6, pp. 1083–1092, Jun. 1999.

[22] P. C. Wang, C. T. Chan, S. C. Hu, C. L. Lee, and W. C. Tseng, "High-speed packet classification for differentiated services in next-generation networks," *IEEE Trans. Multimedia*, vol. 6, no. 6, pp. 925–935, Dec. 2004.

[23] P. Warkhede, S. Suri, and G. Varghese, "Fast packet classification for two-dimensional conflict-free filters," in *Proc. IEEE INFOCOM*, Mar. 2001, pp. 1434–1443.

[24] F. Baboescu and G. Varghese, "Fast and scalable conflict detection for packet classifiers," *Comput. Netw.*, vol. 42, no. 6, pp. 715–735, Aug. 2003.

[25] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small forwarding tables for fast routing lookups," in *Proc. ACM SIGCOMM*, Sep. 1997, pp. 3–14.

[26] Y. Afek, O. Ben-Shalom, and A. Bremler-Barr, "On the structure and application of BGP policy atoms," in *Proc. 2nd ACM SIGCOMM Workshop on Internet Measurement*, 2002, pp. 209–214.

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

[28] D. E. Taylor and J. S. Turner, "ClassBench: A packet classification benchmark," in *Proc. IEEE INFOCOM*, Mar. 2005.

[29] S. Bradner, "Next generation routers. Overview," *Networld Interop.*, 1997.

[30] P. C. Wang, C. T. Chan, and Y. C. Chen, "A fast IP routing lookup scheme," *IEEE Commun. Lett.*, vol. 5, pp. 125–127, Mar. 2001.

[31] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood, "Fast hash table lookup using extended Bloom filter: An aid to network processing," in *Proc. ACM SIGCOMM*, Sep. 2005, pp. 181–192.

[32] Abilene NetFlow Nightly Rep., 2002, [Online]. Available: http://www.itec.oar.net/abilene-netflow/.

[33] Abilene Network, 2004, [Online]. Available: http://abilene.internet2.edu/.

[34] NLANR Project Nat. Lab. Appl. Netw. Res., 2004, [Online]. Available: http://www.nlanr.net

[35] S. Singh and F. Baboescu, Packet Classification Repository, 2004, [Online]. Available: http://ial.ucsd.edu/classification.

[36] P. Gupta, Research Interests, Talks and Publications, 2004, [Online]. Available: http://klamath.stanford.edu/ pankaj/research.html.

Pi-Chung Wang (M'02) received the M.S. and Ph.D. degrees in computer science and information engineering from the National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1997 and 2001, respectively.

From 2002 to 2006, he was with the Telecommunication Laboratories of Chunghwa Telecom Co. Ltd., working on network planning in broadband-access networks and PSTN migration. He also worked on Internet Protocol lookup and classification algorithms. Since 2006, he has been an Assistant Professor of Computer Science at National Chung Hsing University. His research interests include IP lookup and classification algorithms, scheduling algorithms, congestion control, network processors, algorithms, and applications-related computational geometry. He is currently working on high speed string matching for network intrusion detection.

Dr. Wang is a member of the ACM.

Chun-Liang Lee received the M.S. and Ph.D. degrees in computer science and information engineering from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1997 and 2001, respectively.

From 2002 to 2006, he was with the Telecommunication Laboratories, Chunghwa Telecom Co. Ltd. Since 2006, he has been an Assistant Professor of Computer Science and Information Engineering at Chang Gung University, Taoyuan. His research interests include the design and analysis of network protocols, quality of service in the Internet, and packet classification algorithms.

Chia-Tai Chan received the Ph.D. degree in computer science and information engineering from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1998.

From 1999 to 2005, he was a Project Researcher with Telecommunication Laboratories, Chunghwa Telecom Co. Ltd. In 2005, he joined the faculty of the Institute of Biomedical Engineering, National Yang-Ming University, Taipei, as an Associate Professor. His research interests include the design, analysis, and traffic engineering of broadband multiservice networks.

Hung-Yi Chang was born in Taiwan, R.O.C., in 1970. He received the M.S. and Ph.D. degrees in computer science and information engineering from National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., in 1994 and 1999, respectively.

Currently, he is an Assistant Professor in the Department of Information Management with National Kaohsiung First University of Science and Technology, Kaohsiung. His research interests include network management and interconnection networks.