Enhanced prefix inclusion coding filter-encoding algorithm for packet classification with ternary content addressable memory

D. Pao, P. Zhou, B. Liu and X. Zhang

Abstract: Filter encoding can effectively enhance the efficiency of ternary content addressable memory (TCAM)-based packet classification. It can minimise the range expansion problem, reduce the TCAM space requirement and improve the lookup rate for IPv6. However, additional complexity will incur inevitably in the filter table update operations. Although the average update cost of the prefix inclusion coding (PIC) scheme is very low, the worst-case update cost can be significantly higher. Major modifications to the PIC scheme to improve its update performance are presented. The new coding scheme is called PIC with segmented domain. By dividing the field value domain into multiple segments, the mapping of field values to code points can be more structural and help avoid massive code-point relocation in the event of new insertions. Moreover, the simplified codeword lookup for the address fields can be implemented with embedded SRAM rather than with TCAM. Consequently, the lookup rate of the search engine can be improved to handle the OC-768 line rate.

1 Introduction

As the Internet is evolving towards multimedia-oriented and quality-of-service aware applications, more advanced packet processing is necessary for the next generation Internet routers. Advanced routers that are capable of classifying packets into flows are called flow-aware routers [1]. A flow is usually defined by a five-tuple filter consisting of the source and destination address prefix, source and destination port range and the protocol field. Three types of matches, namely, prefix match, range match and exact match can be specified. Usually, the address fields require prefix match, the port number fields require range match and the protocol field requires exact match. Many high-speed packet classification methods have been published in the literature [1, 2]. These methods can be broadly divided into two categories, namely, algorithmic approaches based on embedded SRAM with dedicated hardware devices [3–9] and approaches based on ternary content addressable memory (TCAM) [10–17]. The packet classification problem can be modelled as the classical point location problem in the computational geometry. Let N be the number of regions (filters) and d be the number of dimensions. It is well known that the point location problem can be solved in O(log N) time with O(N^d) space; or, in O(log N^{d-1}) time with O(N) space. In general, algorithmic approaches try to exploit the distribution of field values in the filter table in order to optimise the classification rate and minimise the memory required. As a result, incremental updates to the data structures are rather difficult. To accommodate new rules, the data structures of the search engine need to be rebuilt; or, the performance of the system will be degraded substantially. Because of the flexibility and simplicity in the lookup table management, TCAM has been widely used in commercial routers [18] for the implementation of various lookup functions, such as label lookup in MPLS, IP address lookup and packet classification. According to [19], over 6 million TCAM devices were deployed worldwide in 2004. It is expected that TCAM will remain as the dominant solution that will be used by the industry in the foreseeable future. In this article, we focus on the studies of TCAM-based packet classification.

TCAM allows three possible values to be stored in a memory cell, that is, 0, 1 or x (don't care). Fig. 1 depicts the organisation of a typical TCAM-based lookup engine. A filter is stored in one TCAM entry. The input search key is compared with all the filters in parallel. There is a match bit associated with each TCAM entry. The match bit is set if the search key matches the filter. The priority encoder built-in the TCAM will then return the lowest address of the matching entries, and the corresponding action is retrieved from the SRAM/DRAM.

Top-of-the-line TCAM device available today can operate at 266 MHz with a capacity of 18 Mb [18]. There are 72 I/O pins for entering the search key. The word length is configurable to have 36, 72, 144, 288 and 576 bits. The device supports 133 million searches per second (MSPS) for a 72-/144-bit search key. The lookup rate is reduced to 66.5 MSPS for a 288-bit search key and 33.25 MSPS for a 576-bit search key. When TCAM is applied to packet classification, a range value is converted to multiple prefixes or entries with appropriate don't care bits. For example, the popular port range [1024–65 535] will be decomposed into six distinct prefixes. If both the

doi:10.1049/iet-cdt:20060226

Paper first received 12th December 2006 and in revised form 13th May 2007

D. Pao and P. Zhou are with the Department of Electronic Engineering, City University of Hong Kong, Hong Kong

B. Liu is with the Department of Computer Science and Technology, Tsinghua University, Beijing, People's Republic of China

X. Zhang is with the Department of Computer Science, Carnegie Mellon University, USA

E-mail: d.pao@cityu.edu.hk

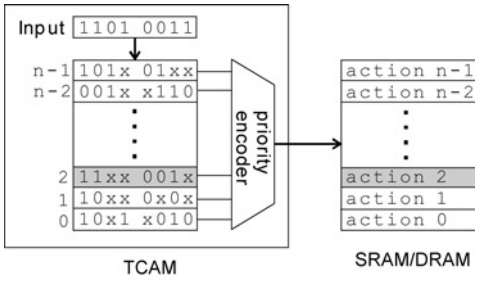

Fig. 1 *Organisation of TCAM-based lookup engine*

source and destination port ranges are specified as ≥ 1024, the given filter will occupy 36 TCAM entries. This is known as the range expansion problem. Previous studies have found that the number of physical entries is two to six times the number of rules in a filter table [14].

High cost and high power consumption are the two major disadvantages of TCAM-based lookup engines. An 18 Mb TCAM costs more than US 200 and consumes up to 18 W of power. A five-tuple IPv4 filter has 104 bits in length. The filter length increases to 296 bits in IPv6. In a straightforward implementation of IPv6 packet classification, a 576-bit word length is selected. Hence, an 18 Mb TCAM can store only a filter table with about 10 K rules, and the lookup rate is reduced to 33.25 MSPS, which is much lower than the peak packet arrival rate of 78 MPPS for an OC-768 data link at 40 Gbps (assume 64-byte IPv6 packets).

Filter encoding [13, 15] is an effective approach to enhance the efficiency of TCAM-based lookup engines in terms of space, time and power dissipation. However, once the fields are encoded, additional complexities incur inevitably in the update process. Traditionally, rules are updated manually and the update frequency is low. However, filter tables are becoming more dynamic. Newer systems are required to provide fast responses to hostile network events, and automatic updates to the filter tables are possible. Packet classification also finds application in the implementation of content-based switches. A packet classification engine is used to support packet header translation for TCP splicing at wire speed [20]. The filter table needs to be updated frequently, as HTTP/SSL (secure socket layer) sessions are established and/or closed down.

In this article, we shall present major extensions to our previously proposed encoding scheme, called prefix inclusion coding (PIC) [13] such that the worst-case update cost to the filter table is substantially reduced. Furthermore, the packet processing rate can be improved by simplifying the codeword lookup operation. The authors of [16, 17] proposed chip-level parallel processing techniques to increase the system throughput by up to four times that of a single chip. These parallel processing techniques can also be applied to encoded filter tables if required. However, the discussion of chip-level parallel classification engine is beyond the scope of this paper.

2 Related works

2.1 Range expansion problem

The work of Liu [12] was one of the early proposals to tackle the range expansion problem using encoding scheme. His scheme uses 1 bit in the codeword to represent a non-trivial range (range that contains more than one point). If a 144-bit TCAM entry is used to store a 104-bit IPv4 filter, up to 40 arbitrary ranges can be encoded. Since the number of arbitrary ranges can be much larger than 40, the effectiveness of this approach is limited. Che *et al.* [21] proposed an algorithm to dynamically select an optimal subset of ranges to be encoded on the basis of Liu's approach so that the expansion factor could be minimised. Lakshminarayanan *et al.* [10] proposed another encoding scheme that would reduce range expansion by expanding the port range field. The 16-bit port range field is split into l chunks of sizes $k_0, k_1 \ldots, k_{l-1}$. Each chunk of k_i bits is then encoded using $2^{k_i} - 1$ bits. These encoding schemes may not be applicable to IPv6. For IPv6 packet classification, it is not economical to use a 576-bit TCAM entry to store a 296-bit filter. Hence, the general idea is to compress the filter length to fit into shorter word length, for example, 72 or 144 bits, in order to reduce the TCAM space requirement rather than expanding the size of individual fields.

Spitznagel *et al.* [14] proposed to incorporate range comparators in each TCAM entry. The range comparator is an iterative circuit that requires 32-gate delay to determine whether the 16-bit port number matches the given range. In today's TCAM cell design [22], whether an entry matches the input search key can be determined in one single-gate delay. Power dissipation in a search operation is mainly due to the pre-charging and discharging of the match lines. Sophisticated current racing technique is employed to reduce the voltage swing of the match line in order to reduce the power dissipation during a search operation. There is a very stringent requirement on the processing time of individual TCAM cell. The excessive long delay time of the range comparator circuit proposed by Spitznagel *et al.* [14] will have negative impacts on the processing time and power dissipation of the TCAM. How could their proposal be incorporated in the physical implementation of TCAM cell array requires further studies.

2.2 P^2C-encoding scheme

The work of van Lunteren and Engbersen [15] was the first encoding method that aimed at compressing the overall filter length. Their method, called P^2C, is based on the concept of primitive-range hierarchy. Ranges are mapped to different layers in the hierarchy such that ranges on the same layer must be disjoint and are represented by non-zero codewords. By default, smaller ranges are placed on the top of larger ranges. Three coding styles are presented in [15]. We shall use the sample prefixes shown in Table 1 to illustrate the idea. In style I, primitive ranges in a layer are assigned distinct non-zero codewords as shown in Fig. 2. The codewords assigned to individual ranges are independent.

P^2C style II allows two primitive ranges on the same layer to be assigned a common code value if both ranges are subranges of two disjoint primitive ranges at other layers. For example, ranges B, E and G are mapped to L2 as shown in Fig. 2. They can be assigned the same code value ('01' in the example) since they are subranges of A, D and F, respectively. The number of layers in the primitive range hierarchy is limited by the physical implementation of the codeword translation hardware. In P^2C style III, primitive ranges on different layers are merged if the number of layers exceeds the limit; or, the length of the composite codeword exceeds the physical limit. However, layer merging may result in filter replication (because a range is being represented by multiple patterns). In this example, prefixes B and H are represented by two and three codewords, respectively.

Table 1: Sample prefixes and codeword assignment using P²C

Prefix	P²C-I	P²C-II	P²C-III
A: 39.50.0.0/16	01xxxxx	01xxxx	01xxx
B: 39.50.64.0/24	xx001xx	0101xx	01001, 0101x
C: 39.50.64.128/28	xxxxx01	010101	01010
D: 98.133.0.0/16	10xxxxx	10xxxx	10xxx
E: 98.133.116.0/24	xx101xx	1001xx	10001
F: 98.176.0.0/16	11xxxxx	11xxxx	11xxx
G: 98.176.8.0/24	xx011xx	1101xx	11001
H: 98.176.58.0/24	xx100xx	1110xx	1101x, 1110x, 11110
I: 98.176.58.16/30	xxxxx10	111001	11011
J: 98.176.58.128/30	xxxxx11	111010	11101

A certain number of spare bits are reserved to support the dynamic addition of new prefixes/ranges. The spare bits can be allocated to any primitive layers on demand. In principle, P²C style I allows efficient incremental updates to the filter table since the codewords assigned to individual prefixes/ranges are independent. However, when the code length of a layer is changed, the codewords for all ranges mapped to that layer need to be modified. Spare bits in the TCAM should be initialised to x (don't care) instead of 0 as suggested in [15]; otherwise it will induce dependencies of the TCAM contents among overlapping ranges mapped onto different layers. Consequently, the complexity of insertion and deletion operations will be substantially increased. For example, if one more range is mapped to layer 1 and the code length for layer 1 is increased to three bits, then the codewords of A, D and F need to be updated. As a result, all the filters associated with prefixes A, D and F also need to be updated.

The update complexity of P²C styles II and III is even higher because of the dependency of codeword assignment caused by the mapping of ranges to layers. For example, if ranges A and D are removed, then B and E cannot share the same code value. Whenever a range r is inserted or deleted, all the ranges enclosed by r would have their codewords modified. For example, if a prefix $K = 98.0.0.0/8$ is added, then the codewords of D, E, F, G, H, I and J are affected. Moreover, the removal of a range r may also affect other ranges not covered by r due to code value sharing.

Li *et al.* [11] proposed a modified encoding algorithm for IPv6 on the basis of P²C. They observed that there are only a few distinct patterns in the first 16 bits of IPv6 unicast addresses. Hence, they proposed to encode only the first 16 bits of the address fields by fixed-size 8-bit codewords. For the port range fields, trivial ranges (ranges covering a single point) and non-trivial ranges are mapped to specific layers. To facilitate incremental updating, the code length of each layer is predetermined and P²C style I is used. The overall length of an encoded filter is 288 bits. A drawback of using 288-bit word length is that the lookup rate of the TCAM device is reduced to 66.5 MSPS. There is also a potential risk for the needs to change to style II or style III encoding in case the distribution of port ranges cannot be accommodated by the predefined number of layers and/or the predefined code length for that layer. This scheme may have good update efficiency at the expense of having slower lookup rate and higher TCAM space requirement.

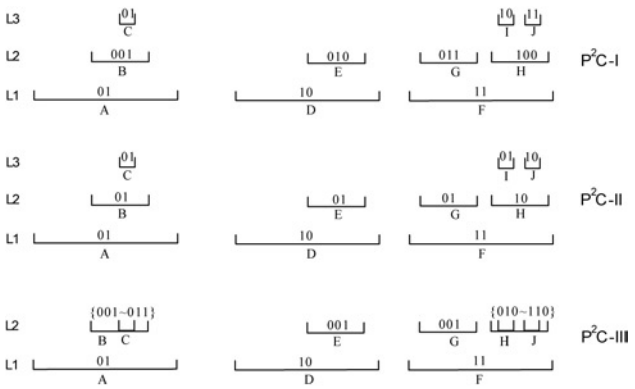

Fig. 2 *P²C codeword assignment*
L3 is merged with L2 in P²C-III

2.3 PIC-encoding scheme

The PIC scheme [13] is based on the inclusion property among prefixes/ranges. Let p and q be two distinct prefixes/ranges where the length of p is shorter than or equal to the length of q. Let C_p and C_q be the codewords assigned to p and q, respectively. The codeword assignment satisfies the following three requirements:

(i) valid codeword must have a non-zero value.
(ii) The inclusion property is preserved. C_q is enclosed by C_p iff q is enclosed by p.
(iii) If q is enclosed by p, then C_q must have a non-zero suffix extension from C_p

Fig. 3 depicts the inclusion tree (i-tree) for the set of prefixes listed in Table 1. The codeword assignment algorithm is based on the codespace allocated to nodes of the i-tree. A leaf node in the i-tree can be assigned a full-length codeword, hence the code space occupied by a leaf node is equal to 1. Since the codeword assigned to a child node must have a non-zero suffix extension, the code space occupied by an internal node is greater than or equal to 1 plus the sum of the code space of its children. Wildcards are allowed only towards the right-hand side of a codeword, hence the code space occupied by an internal node must be a power of 2. The minimum codeword length is then equal to log of the code space occupied by the root. In the following basic codeword assignment algorithm child nodes of the same parent are ordered by the required code space in descending order. The codeword assignment starts with the child node requiring the largest amount of code space (Fig. 4).

The incremental update cost of PIC is obviously lower than that of P²C. First, in PIC, the removal of a range will not affect the codeword assigned to any other ranges, whereas in P²C, both insertion and deletion of ranges may

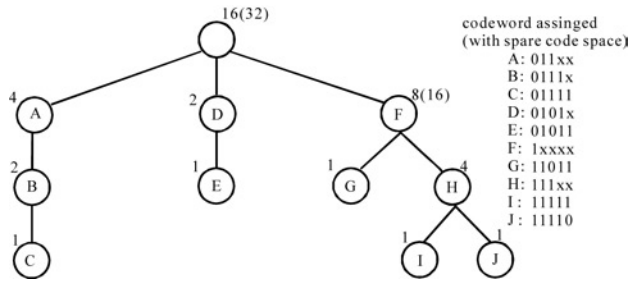

Fig. 3 *Inclusion tree of the sample prefix set*
Number next to a node is the code space allocated
Two times the required code spaces, (the number in bracket), are allocated to node F and the root

```
/* PIC codeword assignment algorithm
  A tree node has 3 attributes:
     1.  childList[] = list of children ordered by required code space in descending order;
     2.  code space allocated to the node;
     3.  codeword assigned to the node.
  Initially the assignCodeword() function is called with t = root, and the codeword assigned to the root is equal to
  the empty string.
*/
assignCodeword(treeNode t)
{    availableCodeSpace = t.codespace;
     for(i = 0; i < t.childList.length; i++)
     {    x = t.childList[i];
          assign a code block of size x.codespace from the higher end of availableCodeSpace to x;
          availableCodeSpace = x.codespace;
          x.codeword = t.codeword + prefix corresponds to the code block assigned in above step;
          assignCodeword(x);
     }
}
```

Fig. 4 *PIC codeword assignment algorithm*

require codeword re-assignment. Secondly, the insertion of a range r in PIC will affect only some (instead of all) of the ranges covered by r. In P^{2}C, the code length of a layer may be expanded due to insertions. When this happens, all the ranges mapped to the given layer will be affected. Thirdly, in P^{2}C, the insertion of a new range may trigger layer merging, which will induce codeword modifications and rule replications. To facilitate dynamic insertions in PIC, extra code space can be pre-allocated to internal nodes of the i-tree. In the example shown in Fig. 3, the code space allocated to node F and the root is two times the required value. It has been shown in [13] that the average cost for dynamic insertions can be very low, only a few TCAM entries are modified for each insertion. However, there exist some worst-case scenarios where the update cost can be much higher. Consider the insertion of a new prefix $K = 98.0.0.0/8$, the structure of the i-tree will need to be adjusted as shown in Fig. 5. The codewords of prefixes D, E and F are modified.

The general reason for the need of large–scale codewords re-assignment is that the nodes (prefixes) are mapped to the code space by the order of the amount of the required code space such that the minimum–length codeword can be derived. When a new range that encloses some existing ranges is added, relocation of code points may be necessary in order to preserve the inclusion relation. We call this the code-point relocation problem, and it is shown in Fig. 6. The address space and code space can be represented by number lines. Because of the insertion of the prefix K, the code points assigned to D and E need to be relocated in order to preserve the inclusion property. In Section 4, it will be shown that most of the prefixes in a filter table are longer than 16 bits. The insertion of 8-bit (or even shorter) prefixes may lead to massive code-point relocation.

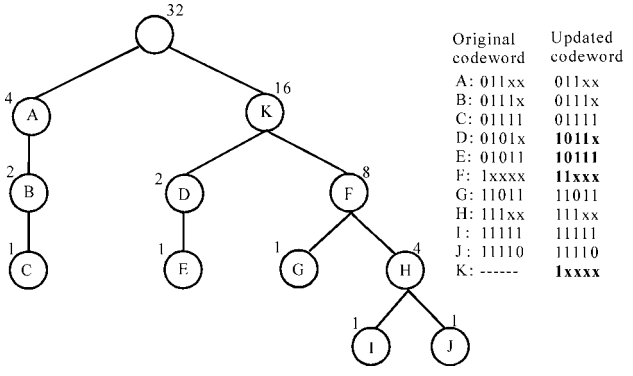

Fig. 5 *Adjustments to the i-tree caused by the insertion a new prefix $K = 98.0.0.0/8$*

3 Enhanced PIC-encoding scheme

Besides the expensive worst-case update cost, there are other practical issues of PIC that worth further studies. It was suggested in [13] that the code tables for the two address fields are implemented using TCAM. This approach has the advantage that a pure TCAM-based classifier can be built without the need of other special-purpose hardware. Hence, it can easily be adopted in existing routers by updating the management software. Three TCAM accesses are required for every classification, and up to five TCAM accesses are required for IPv6 if the two-level codeword lookup approach is adopted. Since the size of today's filter table is only up to about 10 K rules, the encoded filter table and the two code tables can be stored in one single TCAM. Consequently, the lookup rate is only fast enough to handle OC-192 data link at 10 Gbps. This processing rate is sufficient for mid-range routers but not for the most advanced routers with OC-768 data links at 40 Gbps. A straightforward approach to improve the system throughput is to use three separate TCAMs for codeword lookup and classification. But this would be rather expensive.

Another implementation issue is related to the overall code length. It has been found that the minimum overall code length required for large filter tables is between 35 and 38 bits. When the spare bits and control bits for dynamic updates are included, the optimal TCAM word length will be about 48 bits. However, most of today's TCAMs available in the market support word lengths that are multiples of 36. If 72-bit TCAM word length is to be used, it is possible to refine PIC to improve its performance in the lookup speed and incremental updating. In the following subsection, we shall first present our analysis of prefix and port range distribution. On the basis of the observations of the field value distribution, we shall derive an enhanced coding scheme called PIC with segmented domain (PIC-SD).

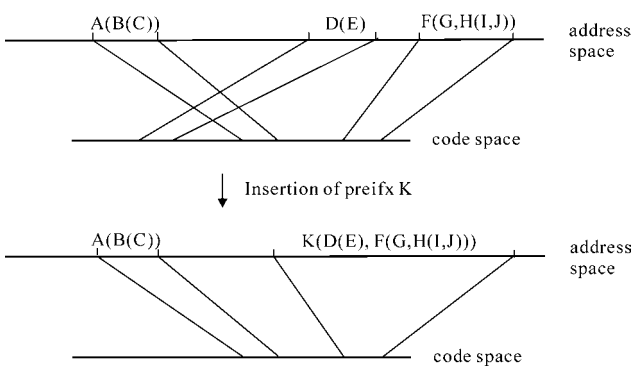

Fig. 6 *Code-point relocation*

Table 2: Distribution of source prefixes by length

Prefix length, bit	acl1 (756)	acl2 (658)	acl3 (2382)	acl4 (2968)	acl5 (3058)	ipc1 (1818)	fw1 (262)	fw4 (275)
1–7	0	0	0	0	0	0	0	0
8	1	3	0	0	6	0	1	1
9–15	0	13	55	43	0	2	1	0
16	0	19	31	20	16	17	2	0
17–23	2	5	11	88	72	75	5	3
24	0	36	72	44	21	56	0	5
25–32	138	95	351	211	89	164	41	35

Size of the filer table is shown in brackets with the name of the filter table

Table 3: Distribution of destination prefixes by length

prefix length, bit	acl1 (756)	acl2 (658)	acl3 (2382)	acl4 (2968)	acl5 (3058)	ipc1 (1818)	fw1 (262)	fw4 (275)
1–7	6	0	0	0	0	0	0	0
8	3	4	0	12	0	0	0	1
9–15	1	24	0	16	0	0	0	0
16	9	37	6	2	0	51	0	0
17–23	24	6	6	9	0	46	4	14
24	13	58	24	33	0	122	0	40
25–32	147	101	238	649	650	210	38	38

3.1 Prefix and port range distribution

Real-life filter tables contain confidential information and are not available in the public domain. Hence, our analysis is based on synthetic filter tables generated using the rule set generator developed by Taylor and Turner [23]. The generation of the synthetic rule set is guided by a seed file derived from the real-life filter table such that the field value distribution will resemble the real-life situation. A total of 12 seed files can be found in Taylor's web site. The original size of some of the seed files is relatively small, for example, with less than 200 rules. These relatively small filter tables are not included in our studies. Tables 2 and 3 summarise the prefix length distribution of the eight filter tables. From these two tables one can observe that 8-bit prefixes are found in most of the filter tables. In one extreme case, 1-bit destination prefixes (0* and 1*) are found in acl1. On the basis of these findings, we conclude that the encoding scheme should be prepared to handle the insertion of short prefixes. In the P^2C method, the insertion or deletion of a 1-bit prefix will likely affect the codewords of half of the prefixes, that is, half of the filters in the table need to be updated.

The user application is represented by the source/destination port number in TCP/IP. Port numbers in the range 0–1023 are assigned to well-known applications such as ftp, web browser, telnet and so on whereas port numbers in the range 1024–65 535 are dynamically allocated by the operating system. The port range 1024–65 535 is commonly found in filter tables. The distribution of port ranges is summarised in Table 4. One interesting observation is that almost all of the filters in the ACL filter tables have the source port number field specified as 1024–65 535 or simply don't care. On the other hand, ACL filter tables have a much larger number of distinct destination port ranges compared with the other two types of filter tables.

It can be seen from Table 4 that almost half of the arbitrary port ranges within 1024–65 535 are distributed towards the lower end, that is within 1024–4999. This can be explained by the fact that some of the port ranges near the lower end are being used by emerging applications, for example, port numbers 1088–1089 are used by Java RMI. Most of the non-trivial ranges found are narrow ranges that span a few points to about a 100 points. Only a few wide ranges other than 1024–65 535 have been found. The values of these wide ranges are (1025–65 535) found in acl1 and acl2;

Table 4: Distribution of destination (source) port ranges (not including the range 1024-65 535)

Port range	0–1023	1024-65 535			
		1024–4999	5000–9999	10 000–29 999	30 000–65 535
acl1	5 (0)	40 (0)	13 (0)	5 (0)	25 (0)
acl2	15 (0)	3 (0)	3 (0)	0 (0)	1 (0)
acl3	38 (1)	60 (0)	42 (0)	16 (0)	9 (0)
acl4	42 (0)	71 (0)	63 (0)	13 (0)	9 (0)
acl5	11 (0)	11 (0)	12 (0)	3 (0)	1 (0)
ipc1	22 (17)	10 (15)	16 (0)	0 (0)	1 (0)
fw1	25 (7)	4 (1)	5 (1)	1 (1)	1 (0)
fw4	22 (12)	3 (3)	7 (5)	8 (1)	0 (0)

Table 5: PIC-SD codeword assignment

Prefix	PIC-SD ($k = 16$)
A: 39.50.0.0/16	0010 0111 0011 0010 xxx
B: 39.50.64.0/24	0010 0111 0011 0010 11x
C: 39.50.64.128/28	0010 0111 0011 0010 111
D: 98.133.0.0/16	0110 0010 1000 0101 xxx
E: 98.133.116.0/24	0110 0010 1000 0101 111
F: 98.176.0.0/16	0110 0010 1010 0000 xxx
G: 98.176.8.0/24	0110 0010 1010 0000 011
H: 98.176.58.0/24	0110 0010 1010 0000 1xx
I: 98.176.58.16/30	0110 0010 1010 0000 111
J: 98.176.58.128/30	0110 0010 1010 0000 110
K: 98.0.0.0/8	0110 0010 xxxx xxxx xxx

The first 16 bits are extracted from the prefix and the last three bits correspond to the PIC codeword

Table 6: Number of non-empty buckets

	Number of non-empty bucket	
Filter table	Source prefix	Destination prefix
acl1	4	82
acl2	20	60
acl3	53	37
acl4	33	122
acl5	19	67
ipc1	34	97
fw1	9	12
fw4	5	10

5001–65 535 found in acl1; 20 001–65 535 found in acl4. The insertion of narrow port ranges can be handled efficiently in PIC. However, the insertion of wide ranges may lead to the code-point relocation problem.

3.2 PIC-SD-encoding scheme

First we shall discuss how to handle the insertion/deletion of short prefixes. We divide the address space into 2^k disjoint segments on the basis of the value of the first Kbits of the IP address. Prefixes longer than k bits are distributed to the corresponding bucket and PIC is applied to encode prefixes in individual buckets separately. The codeword is composed of the first k bits of the prefix followed by the PIC codeword. Prefixes with k or fewer bits need not be encoded. Since most of the prefixes in the filter tables are between 16 and 24 bits, k is assumed to be equal to 16 in the following discussion. Table 5 lists a possible assignment of codewords to the prefix set of Table 1. Using this approach, the insertion/deletion of prefixes with not more than k bits will not affect the codeword assignment of other prefixes.

We analyse the maximum code length of PIC-SD for the eight filter tables with $k = 16$. Table 6 shows the number of non-empty buckets in the eight filter tables, and Table 7 shows the distribution of the required PIC code length. It can be found that out of the 664 non-empty buckets, only one bucket requires 8-bit code length, and nine buckets require 7-bit code length. The rest can be encoded in 6 bits or less. The bucket requiring the longest code length is found in acl1. This is a rather special case where almost 60% of the source prefixes are localised in one bucket. When the filter table grows, it is expected that the prefixes will be spread out to a larger number of buckets.

Using this revised encoding scheme, the codeword lookup process can be largely simplified. Assume the PIC code length is B bits. We can first extract the first 16 bits of the address A, denoted as A_H, to probe into a hash table. If we have a hash miss, then the search key is formed by simply appending B zeros to A_H. If there is a hit in the hash table, a three-level (8-4-4) indexing structure based on the method of [24] can be used to lookup the codeword. We take bits 17–24 of A to access the 256-entry L1 index array. An array entry may store a codeword or a pointer to an index array on the next level. If the selected entry in the L1 index array is a pointer to an L2 index array, then we take bits 25–28 to access the L2 index array. Finally, if the selected L2 index entry is a pointer to an L3 index array, we take bits 29–32 to access the L3 index array. Assume each entry in the multi-level index array has 12 bits. The total amount of memory required to implement the codeword lookup (not including the hash table) is summarised in Table 8. The lookup hardware can be organised as a pipeline to achieve one codeword lookup per cycle.

Next we shall present a heuristic to alleviate the code-point relocation problem for the port range field. In our analysis of port ranges, we find that the basic range $R = 1024–65\,535$ is very popular. In addition, for the ACL filter tables with a larger number of distinct destination port ranges, for example, acl1, acl3, acl4, we find that about 40–50% of the port ranges in R are within 1024–4999, and about 30–40% are within 5000–9999. Two guard ranges 5000–65 535 and 10000–65 535 can be added to the i-tree of the destination port field as shown in Fig. 7. By inserting guard ranges into the i-tree, the mapping of port ranges to code-points is more structural such that the code-point relocation problem can be alleviated. The lower bound of the guard range is elastic (can be adjusted). If there exists

Table 7: Distribution of PIC code length for non-empty buckets

PIC code length							
One bit	Two bits	Three bits	Four bits	Five bits	Six bits	Seven bits	Eight bits
158	115	119	120	108	34	9	1

Table 8: Amount of SRAM (KB) for address field codeword lookup

acl1 (756)	acl2 (658)	acl3 (2382)	acl4 (2968)	acl5 (3058)	ipc1 (1818)	fw1 (262)	fw4 (275)
36.8	36.3	45.4	80.9	39.3	58.6	9.5	7.8

Size of the filter table is shown in bracket

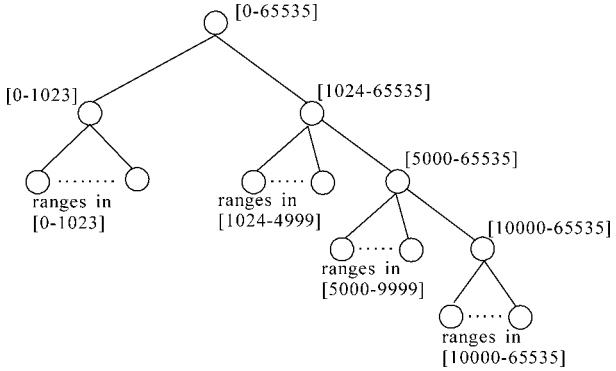

Fig. 7 *Structure of destination port i-tree with guard ranges*

some real port range, say 4800–5100 in the filter table that overlaps with the lower end of the guard range, the lower bound of the guard range will be adjusted to, say, 5100–65 535. Also, if a wide range is to be inserted and the range is close to a guard range, then the inserted range can simply replace the guard range in the *i*-tree. The guard range can even be removed during dynamic updates if necessary. For ACL type of filter tables, there can be 30 or more ranges in 10 000–65 535. Hence, it is recommended that the amount of code space allocated to the guard range (10 000–65 535) be at least 64. Extra code space should also be allocated to the other guard ranges. It is recommended that the code space allocated to 5000–65 535 and 1024–65 535 be at least 256 and 1024, respectively.

4 Performance evaluation

4.1 Code length and TCAM space requirement

Most of today's TCAMs available in the market support word lengths that are multiples of 36. If the length of an encoded filter exceeds 36 bits, we need to store it in a 72-bit TCAM entry. For medium-to-large filter tables, the use of 72-bit word length is necessary. Two control bits are required to support concurrent table lookup and incremental update operations [13]. Hence, we are left with 70 bits to store an encoded filter. Three bits are required to represent the protocol field. If k is set to 16 (prefixes with 16 bits or less are not encoded), and the PIC code length for the address field can have up to 9 bits, we can use 17 bits to encode the two port ranges in which the minimum code length required to encode the two port range fields is equal to 10 bits. The value of k can be adjusted within the range of 12–16, depending on the characteristics of filter tables. Table 9 compares the TCAM space requirements

Table 9: TCAM space requirements

Filter table	Conventional approach, KB	PIC-SD, KB	TCAM space reduction, %
acl1	19.8	6.64	66.5
acl2	21.74	5.78	73.4
acl3	76.57	20.9	72.7
acl4	91.52	26.09	71.5
acl5	72.55	26.88	62.9
ipc1	44.17	15.98	63.8
fw1	16.2	2.3	85.8
fw4	32.95	2.42	92.7

Table 10: Average number of prefixes/ranges affected by the insertion of a new filter

	Source prefix		Destination prefix		Destination port	
	PIC	PIC-SD	PIC	PIC-SD	PIC	PIC-SD
acl1	0.308	0.058	1.068	0.025	0.261	0.105
acl2	0.813	0.085	1.344	0.09	0.049	0.057
acl4*	0.742	0.062	0.6	0.066	0.024	0.013
acl5	0.176	0.013	0.205	0.137	0.003	0.003

of PIC-SD and the conventional approach without encoding, where a filter is stored in a 144-bit TCAM word.

Today's TCAM devices have built-in power management support. The memory array can be partitioned into blocks of 2K × 72 bits. Power consumption in a search operation can be reduced by only enabling a subset of memory blocks in the device rather than searching the whole chip. Space reduction offered by the filter-encoding scheme will certainly help to reduce the overall power consumption.

4.2 Incremental update cost

In this section, we shall compare the update cost of the original PIC scheme and the refined PIC-SD scheme in terms of the number of prefixes/ranges and the number of encoded filters that would be affected by the insertion of a new filter. Readers are referred to [13] for comparisons of PIC with P^2C and for the detailed procedure on how to update the physical TCAM entries without locking the filter table. Since we have two moderate sizes and three large ACL filter tables, our studies will be based on these five ACL filter tables. We use acl3 as the reference table and insert incrementally to it the filters taken from other ACL tables, say acl1. In our experiments, we found that the performance of PIC-SD is quite stable but the worst-case update cost of PIC may have some dependency on the insertion order of the short prefixes. For example, there are several short prefixes in acl1, for example, 1*, 11*, 101*, 01100*. The worst-case update cost can vary significantly when these short prefixes are inserted in different order. Five random insertion traces for acl1 are generated. To have a fair comparison between PIC and PIC-SD, the same insertion traces are applied to the two methods. The evaluation process is repeated for the other three ACL tables, that is, acl2, acl4 and acl5. A total of 20 experiments are performed. At the end of the insertion process, the size of the filter table grows by 28–128%.

Tables 10 and 11 show the average number and maximum number of existing prefixes/ranges affected by the insertion of a new filter. The worst-case value is the largest value out of the five experiments, whereas the average value is the

Table 11: Maximum number of prefixes/ranges affected by the insertion of a new filter

	Source prefix		Destination prefix		Destination port	
	PIC	PIC-SD	PIC	PIC-SD	PIC	PIC-SD
acl1	57	5	438	5	90	18
acl2	49	9	78	6	3	8
acl4*	147	33	115	15	21	5
acl5	69	14	12	8	3	3

Table 12: Average and worst-case number of filters affected by an insertion

	Average case		Worst-case	
	PIC	PIC-SD	PIC	PIC-SD
acl1	6.3	0.64	1790	76
acl2	4.1	0.57	171	99
acl4*	3.1	0.43	321*	244
acl5	1.3	0.39	549	93

* represents an exception condition

overall average of the five experiments. We can see that the average number of existing prefixes/ranges affected by an insertion is very small for PIC, whereas the average of PIC-SD is even smaller. In most of the cases, the insertion of a new rule will not affect the coding of existing prefixes/ranges. On the other hand, PIC-SD has very significant improvement on the worst-case performance. We can see that when the new filters are taken from acl1, more than 400 destination prefixes can be affected by the insertion of a short prefix in the original PIC algorithm. These worst-case scenarios can be avoided in PIC-SD.

A prefix/range can be associated with multiple filters. Hence, the number of filters affected by incremental insertion is higher than the number of prefix/range. In some extreme cases, a prefix/range can be associated with more than 100 filters. If the codewords of some popular prefixes/ranges are modified, a relatively large number (e.g. more than 100) of filters may be affected. Table 12 shows the average and maximum number of filters affected by an insertion. In the original PIC algorithm, code space allocated to an internal node of the *i*-tree can be up to four times the minimum value. The minimum code length of the source and destination address fields of acl3 are 12 and 10 bits, respectively. With the extra code space allocation, the code length for the two address fields are 14 and 12 bits, respectively. There is one important point to note in Tables 10–12. When the system inserts about 1700–2100 rules taken from acl4 to acl3, the source prefix code length grows to 15 bits, that is, the filter table needs to be reorganised. When this exception condition happens, the experiment is stopped. If the experiments were not stopped, the maximum number of filters affected by an insertion for acl4 would be over 3000 (much larger than 321 as shown in Table 12). On the other hand, all the 2968 rules of acl4 can be accommodated using PIC-SD. The required code length of the address field in PIC-SD remains to be equal to 8 bits after all the insertions. However, we would like to emphasise that if the PIC algorithm was allowed to use the same overall code length of PIC-SD (i.e. 70 bits), a lot more spare bits could be preallocated to the address fields and the exception condition should not have occurred.

Table 13 gives a more comprehensive picture of the improvement of PIC-SD over PIC. It shows the number of times out of the 20 experiments when an insertion of a new filter causes 100 or more existing filters to be updated. Actually, there are more than 37 000 insertions performed in the 20 experiments. Only a very small percentage of cases have high update costs, and PIC-SD can effectively avoid most of these expensive cases. There is only one instance out more than 37 000 insertions, in which PIC-SD requires more than 200 filters to be updated. We observe that some of the prefix/port range may have a relatively large number of filters associated with it. If the codeword of a popular prefix/port range is modified, all the associated filters need to be updated.

Table 13: Frequency count of insertions with high update cost

Number of filters affected	100–199	200–299	300–399	400–499	≥500
PIC	106	15	6	7	20
PIC-SD	12	1	0	0	0

5 Extension to IPv6

An IPv6 address has 128 bits. According to the Internet Architecture Board (IAB) and Internet Engineering Steering Group (IESG) recommendation [25], a unicast IPv6 address consists of two components: a 64-bit network/subnetwork ID and a 64-bit host ID. The IAB and IESG [26] recommend that, in general, an address block with a 48-bit prefix be allocated to a subscriber. Very large subscribers could receive a 47-bit prefix or slightly shorter prefix, or multiple 48-bit prefixes. A 64-bit prefix may be allocated when it is known that one and only one subnet is needed; a 128-bit prefix is allocated when it is absolutely known that one and only one device is connecting to the network. It is also recommended 64-bit prefixes be allocated to mobile devices. On the basis of IAB and IESG recommendation, majority of the IPv6 prefixes will have 48–64 bits in length.

We assume the organisation of networks/subnetworks in companies will not be changed substantially when we transit to IPv6. The prefix inclusion properties of IPv6 filter tables should be similar to today's IPv4 filter tables. Hence, an IPv6 filter can be encoded in not more than 72 bits using the original PIC. If PIC-SD is employed, we shall opt for 144-bit TCAM word length. The address space can be segmented using the first 44–48 bits. Assuming that the PIC code length of the address field is 10 bits, we can have at least 26 bits to encode the port number fields and the protocol field.

The address field in a filter specifies either a network/subnetwork or a specific host. We expect the address field is either a prefix with not more than 64 bits or a full-length 128-bit value. The codeword lookup is similar to the case for IPv4, with some minor changes. Assume $k = 48$. Two hash tables, H48 and H128, are maintained by the system. The system will search the two hash tables in parallel. The first 48 bits of address A is used to search table H48, and the full-length address is used to search table H128. If there is a hit in H128, the returned value will be the desired codeword. The search key for looking up the TCAM is formed by appending the codeword to the first 48 bits extracted from address A. If there is a miss in H128 and a hit in H48, we follow the same procedure as in IPv4 to search the associated three-level indexing structure. If we have a miss in both H128 and H48, then the search key is formed by taking the first 48 bits of A and appends to it a string of zeros.

A 296-bit IPv6 filter needs to be stored in a 576-bit TCAM word if it is not encoded. Assume a port range expansion factor of 1.6 for ACL type of filter tables, an IPv6 filter occupies on average 921 bits of TCAM space. If the PIC-SD coding scheme is applied, an IPv6 filter occupies only 144 bits of TCAM space, which is only 15% of the conventional approach without encoding. If the port range-encoding method of Lakshminarayanan *et al.* [10] is applied such that the port range expansion

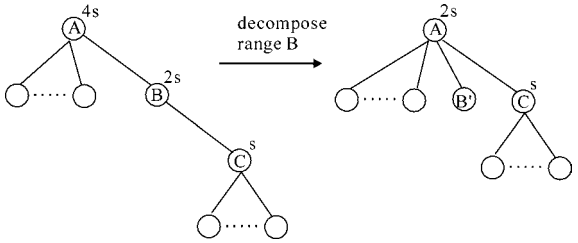

Fig. 8 *Tradeoff between code space utilisation and range decomposition*

problem can be minimised, the TCAM space required by PIC-SD is only 25% of that approach. Besides the reduction of TCAM space requirement, PIC-SD can also increase the lookup rate by a factor of 4.

6 Concluding remarks

The efficiency of TCAM-based lookup engine, in terms of space, time and power, can be improved substantially by filter encoding. However, additional complexities will be incurred in the table update operations. Incremental updating in PIC is easier than P^2C. In P^2C, the subset of ranges affected by an insertion or deletion depends on the inclusion property as well as the mapping of ranges to layers. In PIC, no existing ranges will be affected by deletion. The subset of ranges affected by an insertion can be localised to a subtree in the *i*-tree, and it is easier to manoeuvre the update process to minimise the cost. A major cause of performance deficiency of PIC is the code-point relocation problem because of the insertion of new ranges that encompass a large number of existing ranges. A field domain segmentation approach is proposed to overcome this problem. In the new PIC-SD scheme, field domains are divided into segments. The address field is divided into 2^k disjoint segments using the first k bits of the address value. The port range field is divided into several elastic segments by the introduction of guard ranges. The segmentation of the field domains can allow a more structural mapping of field values to code points. From our experiments, we can see that PIC-SD can reduce the average update cost by more than 80%, and it can avoid almost all of the worst cases in PIC. This is an important advantage of PIC-SD, as the filter tables are expected to be more dynamic.

There can be further refinements to the codeword assignment algorithm in order to enhance the utilisation of code space. Code space utilisation may be low if there are simple branches in the *i*-tree as shown in Fig. 8. One can trade better code space utilisation with range decomposition. By decomposing range $B = B' \cup C$, the code space required by node A can be reduced by 50%, that is the code length of node A can be shortened by 1 bit.

Another advantage of segmenting the address field is that the codeword lookup can be simplified and implemented using SRAM-based methods. The number of TCAM access per classification can be reduced to 1, and the lookup rate is increased. The peak packet arrival rate for OC-768 at 40 Gbps is about 125 MPPS for IPv4 (40-byte packets) and 78 MPPS for IPv6 (64-byte packets). Using the PIC-SD-encoding scheme, lookup engine with a single 266 MHz TCAM capable of 133 MSPS (with 72/144-bit search key) will be able to handle OC-768 line rate for both IPv4 and IPv6 packet classification.

Since most of today's commercial TCAM devices support word lengths that are multiple of 36, the use of 72-bit word length in PIC-SD does not incur additional cost compared with PIC if the length of the original PIC codeword is longer than 36 bits. For the case of IPv6, the use of 144-bit TCAM word length will only increase the overall TCAM space by a small percentage. Because, in PIC-SD, the codeword lookup for the address fields is done using SRAM-based method. The increase in TCAM space for the encoded filter table is offset by the savings in TCAM space for the code tables.

7 References

1 Chao, H.J.: 'Next generation routers', *Proc. IEEE*, 2002, **90**, (9), pp. 1518–1558
2 Gupta, P., and McKeown, N.: 'Algorithms for packet classification', *IEEE Netw.*, 2001, **15**, (2), pp. 24–32
3 Gupta, P., and McKeown, N.: 'Packet classification on multiple fields'. ACM SIGCOMM, 1999, pp. 147–160
4 Gupta, P., and McKeown, N.: 'Classifying packets with hierarchical intelligent cuttings', *IEEE Micro*, 2001, **20**, (1), pp. 34–41
5 Pao, D., and Liu, C.: 'Parallel tree search: an algorithmic approach for multi-field packet classification', *Comput. Commun.*, **30**, (2), pp. 302–314
6 Prakash, A., Kotla, R., Mandal, T., and Aziz, A.: 'A high-performance architecture and BDD-based synthesis methodology for packet classification', *IEEE Trans. Comput.-Aided Des. Int. Circuits Syst.*, 2003, **22**, (6), pp. 698–709
7 Sangireddy, R., and Somani, A.K.: 'High-speed IP routing with binary decision diagrams based hardware address lookup engine', *IEEE J. Sel. Areas Commun.*, 2003, **21**, (4), pp. 513–521
8 Singh, S., Baboescu, F., Varghese, G., and Wang, J.: 'Packet classification using multidimensional cutting'. ACM SIGCOMM'03, 2003, pp. 213–224
9 Taylor, D.E., and Turner, J.S.: 'Scalable packet classification using distributed crossproducting of field labels'. IEEE INFOCOM, 2005
10 Lakshminarayanan, K., Rangarajan, A., and Venkatachary, S.: 'Algorithms for advanced packet classification with ternary CAMs'. ACM SIGCOMM'05, 2005, pp. 193–203
11 Li, W., Xi, Y., Liu, B., and Wang, X.: 'Ultra high-speed IPv6 packet classification using ternary CAMs'. Proc. IEE Irish Signals and Systems Conf., 2005, pp. 129–134
12 Liu, H.: 'Efficient mapping of range classifier into ternary-CAM'. IEEE Symp. on High Performance Interconnects (Hotl'02), 2002
13 Pao, D., Li, Y.K., and Zhou, P.: 'Efficient packet classification using TCAMs', *Comput. Netw.*, 2006, **50**, (18), pp. 3523–3535
14 Spitznagel, E., Taylor, D., and Turner, J.: 'Packet classification using extended TCAMs'. IEEE ICNP, 2003
15 van Lunteren, J., and Engbersen, T.: 'Fast and scalable packet classification', *IEEE J. Sel. Areas Commun.*, 2003, **21**, (4), pp. 560–571
16 Zhang, X., Liu, B., Li, W., Xi, Y., Bermingham, D., and Wang, X.: 'IPv6-oriented 4xOC-768 packet classification with deriving–merging partition and field-variable encoding algorithm'. IEEE INFOCOM, 2006
17 Zheng, K., Che, H., Wang, Z., Liu, B., and Zhang, X.: 'DPPC-RE: TCAM-based distributed parallel packet classification with range encoding', *IEEE Trans. Comput.*, 2006, **55**, pp. 947–961
18 Cypress Semiconductors Co., http://www.cypress.com
19 Bolaria, J., and Gwenmap, L.: 'A guide to search engines and networking memory'. http://www.linleygroup.com/Reports/memory_guide.html, April 2004
20 Apostolopoulos, G., Aubespin, D., Peris, V., Pradhan, P., and Saha, D.: 'Design, implementation and performance of a content-based switch'. IEEE INFOCOM, 2000, pp. 1117–1126
21 Che, H., Wang, Z., Zheng, K., and Liu, B.: 'DRES: dynamic range encoding scheme for TCAM coprocessors'. Technical Report, University of Texas at Arlington, http://crystal.uta.edu/~hche/dres.pdf
22 Arsovski, I., Chandler, T., and Sheikholeslami, A.: 'A ternary content-addressable memory (TCAM) based on 4T static storage and including a current-race sensing scheme', *IEEE J. Solid-State Circuits*, 2003, **38**, (1), pp. 155–158
23 Taylor, D.E., and Turner, J.S.: 'ClassBench: a packet classification benchmark'. IEEE INFOCOM, 2005, http://www.arl.wustl.edu/~det3
24 Gupta, P., Lin, S., and McKeown, N.: 'Routing lookups in hardware at memory access speeds'. IEEE INFOCOM, 1998, pp. 1240–1247
25 Hinden, R., Deering, S., and Nordmark, E.: 'IPv6 global unicast address format'. Network Working Group RCF 3587, August 2003
26 Internet Architecture Board and Internet Engineering Steering Group: 'IAB/IESG recommendations on IPv6 address allocations to sites'. Network Working Group RFC 3177, September 2001