Available online at www.sciencedirect.com
ScienceDirect
Computer Networks 50 (2006) 3523–3535
Computer Networks
www.elsevier.com/locate/comnet

Efficient packet classification using TCAMs

Derek Pao *, Yiu Keung Li, Peng Zhou

Department of Electronic Engineering, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong

Received 21 September 2005; accepted 31 January 2006
Available online 20 March 2006

Responsible Editor: J. Chao

Abstract

Multi-field packet classification is necessary to support advanced Internet functions, such as network security, quality of service provisioning, traffic policing, virtual private networking, etc. *Ternary content addressable memory* (TCAM) is currently the dominant solution method used by the industry because of its speed and the simplicity of filter table management. High cost and high power consumption are the two major drawbacks of TCAM-based lookup engines. Adoption of IPv6 with increased address length will further exacerbate the challenges. In this article, we present a filter encoding method, called *prefix inclusion coding* (PIC) to improve the efficiency of TCAM-based lookup engines. Filters are stored in an encoded format to reduce storage requirement. Codeword assignment in PIC preserves the inclusion relationship among prefixes/ranges. By doing so, a prefix will be represented by a single codeword, and unnecessary filter replication can be avoided. Codeword lookup is equivalent to finding the longest matching prefix in the codeword table. Hence, a pure-TCAM lookup engine can be built without the needs of other semi-custom ASICs in the system. Our method can reduce the TCAM storage requirement by 70% to over 90%. The reduction in TCAM storage requirement also helps to alleviate the high power dissipation problem. The proposed method can be applied to both IPv4 and IPv6.

Keywords: Multi-field packet classification; Ternary content addressable memory; Prefix inclusion coding; Filter compression; IPv4 and IPv6

1. Introduction

The development of the Internet is entering a new era as many countries have started the deployment of IPv6. Accompany with the rapid technological advancement is a new series of challenges. In this article we shall discuss the multi-field packet classification problem which is essential for supporting advanced functions, such as network security, quality of service (QoS) provisioning, traffic policing and control, billing and accounting, and virtual private networks. The following header fields are generally used in the packet classification function: the source and destination address, source and destination port number, and the protocol field. A d-dimensional classification rule (filter) has d components. $F[i]$ is the ith component of the filter F. A packet P is said to match filter F, if $\forall i$, the ith field of the header of

* Corresponding author. Tel.: +852 2788 8607; fax: +852 2788 7791.
E-mail address: d.pao@cityu.edu.hk (D. Pao).

1389-1286/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2006.01.009

P matches $F[i]$. Three types of matches can be specified, namely, *exact* match, *prefix* match and *range* match. In general, the source and destination addresses require prefix match, the source and destination port numbers require range match, and the protocol field requires exact match. A packet may match more than one filter in the classifier. In this case, the *action* associated with the highest priority filter (the first matching filter found in the filter list) will be applied.

A packet may go through several classification steps in a router. It is checked against an access control list upon arrival to determine whether it will be accepted or rejected. This classification function can be used to provide packet filtering and IPSec security associations. A second classification may be used to regulate the traffic on a per-flow basis. This classification function can be used to support QoS provisioning, traffic shaping, and billing and accounting. Packet forwarding can take two paths, i.e. the default next-hop returned by the IP address lookup function, and a more intelligent policy-based forwarding based on the layer 3 and layer 4 header information. Hence, a router may need to maintain multiple filter databases. Today's classifier can have over four thousands filters, and the number of filters is expected to grow in the future. It is a great challenge to design cost effective lookup engines that can (i) meet the stringent requirements on high lookup rate; (ii) possess good scalability to handle large filter databases and 128-bit address fields in IPv6, and (iii) allow incremental updates to the filter tables.

Many sophisticated algorithmic methods for the packet classification problem have been proposed by other researchers. In [6,7,16], a decision tree is built by analyzing the distribution of field values of the filter database so as to optimize the classification rate. In [2,19], individual field lookups are performed, and the best matching filter is then determined by computing the cross-product of the intermediate lookup results. In [2], the intermediate lookup results are represented using bit-vectors and the cross-product is obtained by computing the logical-and of the bit-vectors. In [19] the intermediate lookup results are represented by set of labels, and the cross products are computed on-the-fly. In [14,15], the classification process is formulated using binary decision diagrams (BDDs), and the BDDs are mapped to hard-wired logic using FPGA. A common drawback of the above approaches is that incremental updates to the filter database will either require rebuilding the data structures and/or the logic blocks [2,6,14,15], or lead to performance degradation [7,16,19]. Readers may refer to [4,8] for comprehensive surveys in this topic.

Ternary content addressable memory (TCAM) is currently the dominant method used by the industry [5,9]. TCAM allows three possible values to be stored in a memory cell, i.e. 0, 1, or x (do not care). Fig. 1 depicts the organization of a typical TCAM lookup engine. A filter is stored in one TCAM entry. The input search key is compared to 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.

When TCAM is applied to packet classification, a range value is converted to multiple prefixes or entries with appropriate do not care bits. For example, the popular port range [1024–65,535] will be decomposed into 6 distinct prefixes. If both the source and destination port ranges are specified as $\geqslant 1024$, the given filter will be replicated 36 times. Previous studies have found that the number of physical entries in the filter table is 2–6 times the number of rules in a classifier [17].

High cost and high power consumption are the two major disadvantages of TCAM-based lookup engines. Top-of-the-line TCAM device available today has a capacity of 18 Mb, and consumes up to 18 W of power [5]. The word length is configurable to have 36, 72, 144, 288, and 576 bits. The lookup rate is reduced if a word length of more than 144 bits is selected. A 5-tuple IPv4 filter has 104 bits in length. The filter length increases to 296 bits in IPv6. Taking into account of the decomposition of range values, a 18 Mb TCAM may only be able to

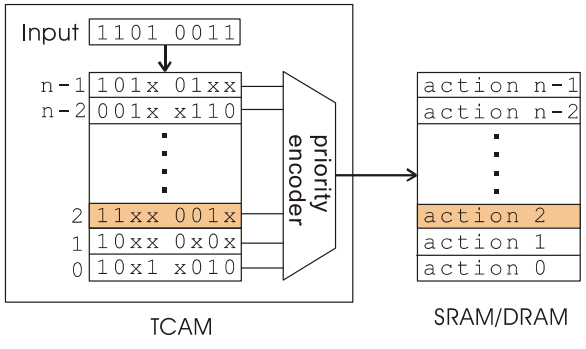


Fig. 1. Organization of a typical TCAM lookup engine.

support an IPv6 filter database with 8–20 K rules. Another drawback of TCAM is the I/O pin limitation. Only 72 I/O pins are available for entering the search key. For IPv6 classification, it takes 5 iterations to enter the 296-bit search key to the TCAM. Hence, the lookup rate is much lower than the peak operating speed of the TCAM.

Liu [10] described a technique to encode range values to reduce filter replication. His scheme uses 1 bit in the codeword to represent a non-trivial range (range that contains more than 1 points). Since the word length of a TCAM entry is limited and there can be over 150 arbitrary ranges found in a filter set [18], the effectiveness of this approach is quite limited. Spitznagel et al. [17] proposed to incorporate range comparators in each TCAM entry. The range comparator is an iterative circuit that requires 32 gates delay to determine if the 16-bit port number matches the given range. In today's TCAM cell design [1] 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 will have negative impacts on the processing time and power consumption of the TCAM. How could their proposal be incorporated in the physical implementation of TCAM cell array requires further studies.

Filter encoding is an effective approach to enhance the efficiency of the TCAM lookup engine. An address field in IPv4 has 32 bits, however, the number of filters in a classifier is much less than 2^{32}. The ratio of the number of filters to the size of the address space is even smaller in IPv6. Hence, it is possible to represent a 32-bit (or 128-bit) address field by a much shorter label or codeword. Similarly, an arbitrary port range can also be represented by a codeword and filter replication can be minimized.

van Lunteren and Engbersen [21] proposed an encoding scheme called P^2C 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 codewords. An address prefix is represented by a composite codeword obtained by concatenating the codeword of individual layers. Three coding styles were presented in [21]. Primitive ranges in a layer are assigned distinct non-zero codewords in style I. Fig. 2 depicts an example of the P^2C coding styles. A prefix is represent by a range on the number line labeled by a single character. By default smaller ranges are placed on top of larger ranges. In the style I encoding scheme, prefixes A, G and I are mapped to layer 1 and they are assigned codewords 01, 10 and 11, respectively. Similarly, prefixes B, E, F, H and J are mapped to layer 2, and prefixes C, D, K and L are mapped to layer 3. Regions on a layer

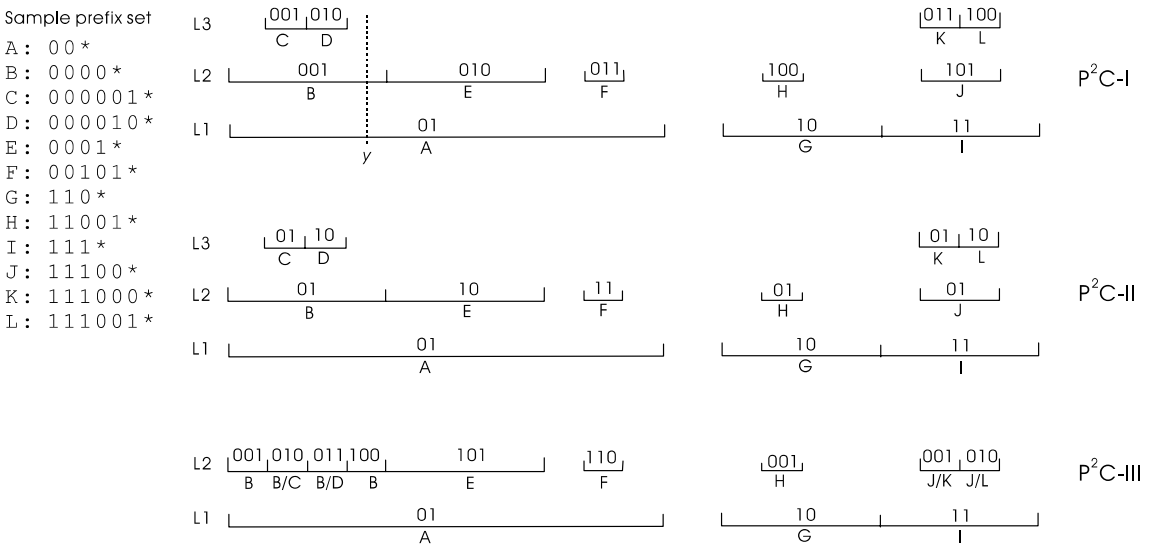


Fig. 2. P^2C coding styles.

Table 1
P^2C codeword assignment

Prefix	P^2C Style I	P^2C Style II	P^2C Style III
A	01******	01****	01***
B	**001***	0101**	01001, 0101*, 01100
C	*****001	010101	01010
D	*****010	010110	01011
E	**010***	0110**	01101
F	**011***	0111**	01110
G	10******	10****	10***
H	**100***	1001**	10001
I	11******	11****	11***
J	**101***	1101**	11001, 11010
K	*****011	110101	11001
L	*****100	110110	11010

not covered by any range are assigned a default codeword of zeros. To perform packet classification, the source and destination addresses are first translated into the corresponding values in the code space using IP address lookup engines such as BART [20]. The codeword lookup method is required to find all matching ranges (on different layers of the primitive range hierarchy) and combine the code segments for each primitive layer to form the final codeword. For example, the codeword corresponds to the input address y marked in coding style I of Fig. 2 is 01 001 000.

P^2C style II allows two primitive ranges on the same layer to be assigned a common codeword if both ranges are subranges of two disjoint primitive ranges at other layers. In the example shown in Fig. 2, prefixes B, H and J on layer 2 can be assigned a common codeword because B is a subrange of A, H is a subrange of G, and J is a subrange of I. By doing so, the overall length of the codeword can be reduced. 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. In the example of Fig. 2 the number of layers in coding style III is set to 2 and the ranges on layer 2 are merged with the ranges on layer 3. Consequently, prefix B is covered by 3 codewords {01001, 0101x, 01100}, i.e. the filter containing prefix B will be replicated 3 times. Table 1 gives the codeword assignment for prefixes shown in Fig. 1 using the 3 coding styles of P^2C. The code length of the 3 styles are 8, 6 and 5, respectively.

In general, the codeword length of styles II and III can be shorter than that of style I. However, incremental updates to the filter table are much more difficult in styles II and III because of the dependencies among overlapping ranges on different layers. During dynamic updates, it is easy to expand the codeword length of a layer to accommodate the new filters. However, it is extremely difficult to compact or reclaim the code space when filters are deleted. Another limitation of the P^2C method is that the scalability of the BART address lookup method to IPv6 with 128-bit address length is unlikely.

In this paper a novel encoding scheme called *prefix inclusion coding* (PIC) scheme is presented. The proposed method can have better compression ratio compared to P^2C. In PIC codeword translation is equivalent to finding the longest matching prefix, hence a pure-TCAM lookup engine can be built. This would eliminate the needs to have semi-custom ASICs in the system and help to reduce the overall system cost for medium sized classifiers with a few thousands filters. Comparing to the conventional TCAM lookup engine, our method can reduce the TCAM space requirement by 70% to over 90%. Incremental updating in the proposed method is easier than P^2C styles II and III. Details of the PIC scheme is presented in Section 2 and performance evaluation will be presented in Section 3. Section 4 discusses the extension to IPv6 and Section 5 is the conclusion.

2. Prefix inclusion coding scheme

A 5-tuple filter $\langle F_1, F_2, F_3, F_4, F_5\rangle$ is stored in the TCAM in an encoded format $\langle C_1, C_2, C_3, C_4, C_5\rangle$ where C_i is the codeword for field F_i. Since the lengths of the codeword are much shorter than the corresponding fields in the original filter, especially the address fields in IPv6, significant savings in TCAM space can be achieved. Individual fields from the packet header are first translated into the corresponding codewords, and the codewords are put together to search the TCAM array.

We shall first describe the *prefix inclusion coding* (PIC) scheme for the source and destination address prefixes. Let p and q be two distinct address prefixes where the length of p is shorter than or equal to the length of q. Prefixes p and q are either disjoint, or q is enclosed by p (i.e. q is a subrange of p). Let C_p and C_q be the codewords assigned to p and q, respectively. The codeword assignment satisfies the following three requirements:

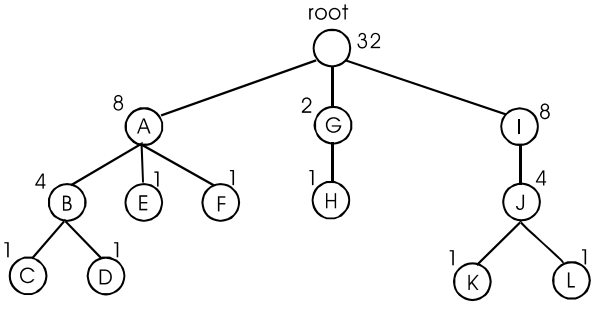
(i) A valid codeword must have a non-zero value.
(ii) The inclusion property is preserved. C_q is enclosed by C_p iff q is enclosed by p.
(iii) If q is enclosed by p, then C_q must have a non-zero suffix extension from C_p.

The system maintains two separate codeword tables storing the $\langle prefix, codeword\rangle$ pairs for the source and destination prefixes, respectively. An input address A is first translated to a codeword C_A by finding the longest matching prefix in the corresponding codeword table. If A matches prefixes p and q, then C_A matches codewords C_p and C_q.

Fig. 3 depicts an example of PIC using the sample set of prefixes in Fig. 2. The set of prefixes are sorted in ascending order (based on lexicographical order). Given a sorted list of n prefixes, a tree (i-tree) that represents the inclusion relationship can be constructed in O(hn) time where h is the depth of the tree. Depth of the inclusion tree depends on the maximum nesting level of prefixes. It has been found that the maximum nesting level of prefixes in real-life classifiers is about 6 [3,18]. A leaf node in the i-tree can be assigned a full-length codeword that occupies a single point in the code space. 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. Since wildcards are only allowed towards the right hand side of a codeword, the code space occupied by an internal node must be a power of 2. The minimum codeword length is then equal to $\log_2$ of the code space occupied by the root. In this example, the codeword length is 5 bits. The code space occupied by nodes can be determined in O(n) time by traversing the tree in post-order. Codewords can then be assigned in O(n) time by traversing the tree in pre-order using the recursive algorithm shown below. The total time to order the branches in the i-tree based on the associated code space is O($n\log n$). Hence, the overall computation complexity of PIC is O($n\log n$).

```
/* This function assigns codewords to children of
node t
   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;
      update availableCodeSpace;
      x.codeword = t.codeword + prefix  corre-
      sponds to the code block assigned in above
      step;
      assignCodeword(x);
   }
}
```

An address value is first translated to a value in the code space before the encoded filter table is searched. In PIC the address translation is equivalent to finding the longest matching prefix in the codeword table. In the codeword table, a do not care bit in a codeword is substituted by a zero.



prefixes	codewords
A: 00*	11xxx
B: 0000*	111xx
C: 000001*	11111
D: 000010*	11110
E: 0001*	11011
F: 00101*	11010
G: 110*	0111x
H: 11001*	01111
I: 111*	10xxx
J: 11100*	101xx
K: 111000*	10111
L: 111001*	10110

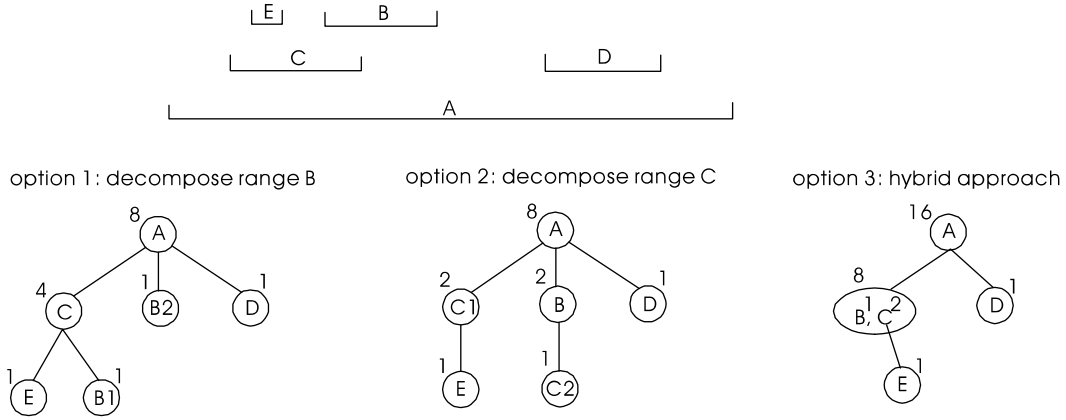
Fig. 3. Inclusion relationship of the sample prefixes and the assigned codewords. The number next to a tree node represents the code space allocated to that node.

For example, if the 8-bit input address is 1110 0111, the best matching prefix is J = 11100* and the returned code value will be 10100 (bits corresponding to the do not care bits in the codeword are underlined). The codeword lookup can be implemented using TCAM or other efficient algorithmic methods [5,13]. For medium size classifiers with about a thousand source/destination prefixes, using TCAM for codeword lookup can be more economical because it eliminates the needs to have other types of semi-custom ASIC devices in the system. The use of algorithmic codeword lookup engine is justified if the number of prefixes is very large.

PIC can also be applied to encode port range values. Port numbers from 1 to 1023 are assigned to well-known applications, such as email, ftp, http, etc., whereas port numbers from 1024 to 65,535 are for other user applications. In the client–server model, the port number used by the server is predetermined, whereas the port number of the client is assigned automatically by the operating system of the client machine within the range [1024–65,535]. Hence, the two port ranges [0–1023] and [1024–65,535] are commonly found in filter sets. Other arbitrary port ranges are, in general, subranges of these two ranges.

In the PIC encoding scheme if 2 port ranges are partially overlapping, one of the 2 ranges will be decomposed. Consider the example shown in Fig. 4 where ranges B and C are partially overlapped. Using a pure PIC encoding scheme, there can be 2 options: (1) range B is decomposed into 2 subranges, or (2) range C is decomposed into 2 subranges. If the number of filters with the port number field equal to B is less than the number of filters with port number field equal to C, then range B will be decomposed, otherwise range C will be decomposed.

One may avoid port range decomposition by using a hybrid encoding method that combines PIC with P^2C. In the hybrid method, partially overlapped ranges are grouped into a composite range. If the hybrid encoding method is used, ranges B and C are combined to form a composite range BC in the *i*-tree. Within the composite range, B and C are encoded using P^2C style II, i.e. B is assigned a code value of 1x, and C is assigned a code value of x1. If *k* bits are used to encode individual ranges within the composite range, the code space allocated to the composite node in the *i*-tree is equal to 2^k times the maximum code space required by individual components of the composite range. In this example, the code space allocated to the composite node BC is equal to 8. The code length of the hybrid method is 1 bit longer than the pure PIC schemes, but port range decomposition can be avoided. If there is a range *R* that spans the two basic ranges [0–1023] and [1024–65,535], then it is



range	option 1	option 2	option 3
A	1xxx	1xxx	1xxxx
B	1110, 1011	110x	111x1
C	11xx	111x, 1101	11x1x
D	1010	1011	10111
E	1111	1111	11x11

Fig. 4. Encoding partially overlapped ranges. Code block corresponds to the composite range BC (option 3) is underlined.

not recommended to use the hybrid method to encode the three ranges. It would be more effective to decompose R into two subranges.

Since the port number only has 16 bits, the codeword translation can be implemented using direct table lookup with a 64 K-entry array. For the protocol field, the most popular values are TCP and UDP. In real-life classifiers, the protocol filed can be represented by a 3-bit or 4-bit codeword.

3. Performance evaluation

Real-life filter databases contain confidential information and are not available in the public domain. We use the rule set generator developed by Taylor [18] to generate sample filter sets for evaluation. Generation of synthetic rule set is guided by a parameter file derived from real-life classifier. We set the sizes of the synthetic rule sets to be about the same as the original classifiers so that the distribution of field values resembles the original classifiers. Taylor's web site provides 12 parameter files that characterize real-life IPv4 rule sets collected from ISPs, network equipment vendors and researchers working in the field. There are three types of classifiers available in Taylor's web site, namely, the access control list (ACL), firewall (FW) and IP chain (IPC). In our study the number of layers allowed in P^2C style III is set to 3 as suggested in [21].

Table 2 shows the required code length for the five fields in a filter. Results for 4 of the small synthetic rule sets with less than 200 rules are omitted because the encoding method is mainly aimed at medium to large filter sets. In a few cases the code length of the source port range is zero because the source port range is equal to wildcard for all the filters in the rule set. Out of the 12 synthetic classifiers, only two instances of partially overlapping port ranges are found. Hence, in our study a pure PIC encoding scheme is used to encode the port range field. Decomposing one of the port ranges in PIC leads to the replication one filter in each of these two cases. P^2C style III may have better compression ratio than style I and II in some of the rule sets, e.g. acl2, acl3 and acl4. However, extensive rule replications are observed as shown in Table 3. In general, PIC has better performance than P^2C. The PIC codeword assignment algorithm is implemented in J2SE 5.0 running on a 2 GHz P4 processor. For a 3 K rules classifier (e.g. acl4), the CPU time to sort the prefixes and port ranges, construct the inclusion

Table 2
Code length for the source and destination prefix, the source and destination port range, and the protocol field

Rule set (size)	acl1 (756)	acl2 (658)	acl3 (2382)	acl4 (2968)	acl5 (3058)	fw1 (262)	fw4 (275)	ipc1 (1818)
P^2C-I	16/18/0/18/2 (54)	23/25/0/9/3 (60)	35/23/1/22/3 (84)	36/33/1/23/3 (96)	17/18/0/10/3 (48)	13/13/6/9/3 (44)	10/17/9/8/3 (47)	20/32/9/11/3 (75)
P^2C-II	15/15/0/18/2 (50)	16/18/0/9/3 (46)	20/17/1/21/3 (62)	24/30/1/22/3 (80)	12/12/0/10/3 (37)	12/11/6/9/3 (41)	10/13/9/7/3 (42)	14/19/8/11/3 (55)
P^2C-III	15/14/0/14/2 (45)	14/14/0/9/3 (40)	17/17/1/16/3 (54)	17/23/1/18/3 (62))	12/12/0/10/3 (37)	12/11/6/9/3 (41)	10/13/9/7/3 (42)	14/16/8/11/3 (52)
PIC	9/9/0/8/2 (28)	10/10/0/5/3 (28)	12/10/1/9/3 (35)	13/12/1/9/3 (38)	10/10/0/6/3 (29)	7/7/4/6/3 (27)	6/8/6/7/3 (30)	10/11/6/6/3 (36)

The overall length of the codeword is shown in bracket.

Table 3
Number of replicated filters in PIC and P^{2}C-III

Rule set (size)	acl1 (756)	acl2 (658)	acl3 (2382)	acl4 (2968)	acl5 (3058)	fw1 (262)	fw4 (275)	ipc1 (1818)
P^{2}C-III	54	265	598	735	0	0	0	236
PIC	0	0	0	1	0	0	0	1

trees and compute the codewords for the address and port range fields is about 76 ms.

3.1. TCAM space requirement

In the studies of space efficiency, we are mainly concerned about the TCAM storage as SRAM and DRAM are much cheaper than TCAM. For example, the cost of TCAM is about 4 times the cost of SRAM of comparable speed, and the power consumption of TCAM is more than 10 times that of SRAM. Most of the commercial TCAM devices support configurable word length of 36, 72, 144, 288 and 576 bits [5]. Some company provides devices that support word length of 32, 64 and 128 bits, and there are devices that support more flexible word length between 32 and 80 bits [11]. The word length supported by commercial TCAM devices may change over time. Hence, in the following discussion we shall compare the raw space required by PIC and that of the conventional approach where filters are not encoded. Table 4 shows the statistics of the synthetic rule sets. The expansion factor, f, is defined to be the ratio of the number of physical TCAM entries required in the conventional method over the number of filters in the rule set. Let N be the number of filters in a rule set. The length of a 5-tuple IPv4 filter is 104 bits. The amount of TCAM space required by the conventional method is equal to $fN\times$ 104 bits.

In PIC, TCAM space requirement consists of two components, namely the space for storing the encoded filters and the space for the codeword tables for the two address fields. The number of distinct source/destination prefixes in the synthetic rule sets are shown in Table 4. Let n be the total number of distinct prefixes, L be the length of the overall codeword for a 5-tuple filter, and r be the number of replicated entries due to partially overlapping port range. TCAM space for the encoded filter table is equal to $(N+r)\times L$ bits, and the space for the codeword table is $n\times 32$ bits. Table 5 compares the TCAM space requirement of PIC with the conventional method. PIC can reduce the TCAM space requirement by about 70% for ACL/IPC types of classifiers, and about 90% for FW classifiers.

3.2. Incremental updating

Filter table updating is more difficult when the filters are stored in encoded format. In P^{2}C the code length of a layer may be expanded when new ranges are added. In styles II and III, removal of a range may trigger the reassignment of codewords to a large number of other ranges. The system needs to update the codeword lookup engines for the two address fields in addition to the encoded filters stored in the TCAM array. In a dynamic environment, once the code length of a layer is expanded due to insertions, it is very difficult to compact the code length during deletions. Incremental update

Table 4
Synthetic rule sets statistics

Synthetic rule set	No. of filters	No. of source prefixes	No. of destination prefixes	No. of TCAM entries required by the conventional method	Expansion factor
acl1	756	142	206	1126	1.49
acl2	658	172	276	1233	1.87
acl3	2382	620	455	4357	1.83
acl4	2968	407	903	5203	1.75
acl5	3058	205	766	4128	1.34
fw1	262	51	52	922	3.52
fw4	275	45	116	1877	6.83
ipc1	1818	315	622	2510	1.38

Table 5
Comparison of TCAM space requirements

Synthetic rule set	Space required by the conventional method (Kbyte)	PIC		
		Space for codeword tables (Kbyte)	Space for encoded filter table (Kbyte)	Space reduction compared to conventional method (%)
acl1	14.3	1.36	2.58	72.4
acl2	15.7	1.75	2.25	74.5
acl3	55.3	4.2	10.18	74.0
acl4	66.1	5.12	13.77	71.4
acl5	52.4	3.79	10.83	72.1
fw1	11.7	0.4	0.86	89.2
fw4	23.8	0.63	1.01	93.1
ipc1	31.9	3.66	7.99	63.4

procedure for P^2C styles II and III was not discussed in [21].

Code space management in PIC is conceptually similar to the classical blocks packing problem where a given number of variable-sized objects are to be packed into fixed-size boxes. The packing problem can be relatively easy if sufficient spare spaces are available. By allowing one extra bit in the codeword, size of the code space is doubled. To facilitate dynamic insertions, additional code spaces are allocated to internal nodes of the *i*-tree. We shall first compute the minimum code space required using the algorithm described in the previous section. In our evaluation, a node with more than 15 descendants is allocated 4 times the minimum required code space, and a node with 3–15 descendants is allocated 2 times the minimum required code space. By doing so, the length of the codeword for the given field can be increased by 2 bits. When a new node y is added to the *i*-tree, the minimum code space required by nodes along the path from the root to y is recomputed. Let node t be the closest ancestor of y such that the required code space after the insertion does not exceed the originally allocated code space. Then codeword reassignment will be localized to a branch of node t that contains the new node y. Fig. 5 shows an example when a new filter F_M with source prefix $M = 110011^*$ is inserted to the sample prefix set. Extra code spaces are allocated to internal nodes A and I as mentioned previously. In this example, the length of the codeword is increased by 1 bit. After the insertion, the code space required by nodes G and H are increased. Hence, codewords of prefixes G and H are to be modified. Consequently, the field values of the encoded filters F_G and F_H need to be updated.

We use the acl4 filter table as a reference and incrementally insert to it all the rules of acl1 in random order. The experiment is repeated with inserting all the rules of acl2, and 800 rules randomly selected from acl3/acl5. New rules are appended to the end of the filter table. The size of the filter table grows by 22–26% after all the insertions. The costs of incremental insertion (in terms of the number of codewords and encoded filters that need to be

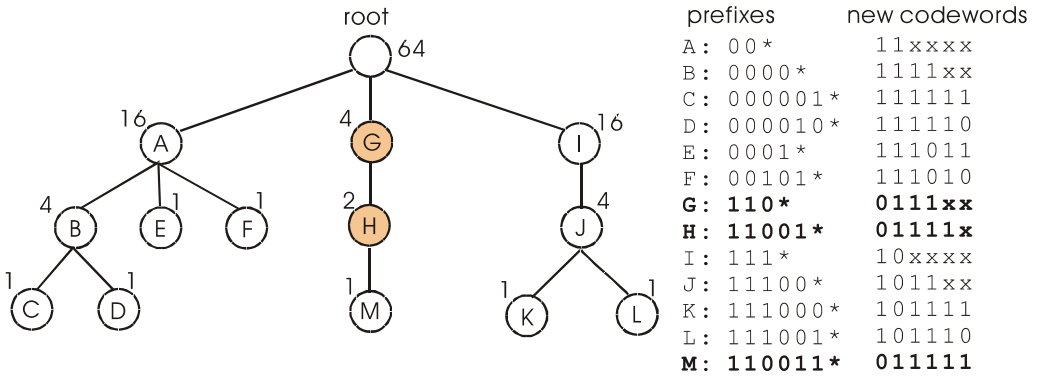


Fig. 5. Effects of inserting a new prefix M on codeword assignment in PIC.

Table 6
Cost of inserting a new filter to the acl4 rule set

Rule set from which new filters are selected	No. of source prefix codeword updated		No. of destination prefix codeword updated		No. of destination port range codeword updated		No. of encoded filters updated	
	Average	Maximum	Average	Maximum	Average	Maximum	Average	Maximum
acl1	1.08	34	1.02	2	1.03	12	1.54	100
acl2	1.26	19	1.39	19	1.02	2	2.36	74
acl3	1.44	23	1.03	6	1.0	2	1.85	73
acl5	1.06	10	1.03	2	1.0	2	1.31	74

updated) are summarized in Table 6. When a prefix is removed, codeword assignment for other prefixes will not be affected.

The insertion process involves modifying the codeword tables and the encoded filter table. Most of the time only a few entries in the tables need to be updated. In those cases, the TCAM lookup engine can be locked during the update process. In the worst case, about 30 entries in the codeword table and 100 encoded filters need to be modified. The update process takes longer time, and locking the lookup engine for a long period of time may not be desirable. The *consistent policy table update* approach (CoPTUA) [22] can be adopted to minimize the impact on packet processing. The general idea is to maintain a consistent view of the lookup tables throughout the update process so that packet classification can be done concurrently (interleaved) with the update process. The process to insert a new filter in PIC can be broken down into five steps:

1. Update the codewords of the affected prefixes. This involves modifying the data stored in the associated SRAM/DRAM of the codeword tables whereas the contents of the TCAM prefix table need not be changed.
2. Update the codeword tables for the port number fields if required. This involves modifying the data stored in the two 64 K-entry arrays.
3. Update the field values of the affected entries in the encoded filter table.
4. Insert the new source/destination prefix into the corresponding codeword table.
5. Insert the new filter to the encoded filter table.

Steps 4 and 5 can be handled by CoPTUA. We shall discuss in more details about steps 1–3. In steps 1 and 2, only the SRAM/DRAM tables need to be modified. The system can maintain a shadow copy of the array. After the changes are made to the shadow copy, the system switches over to use the shadow copy as the working copy. Fig. 6 depicts the changes to the codeword table when a new filter F_M with source prefix $M = 110011^*$ is inserted.

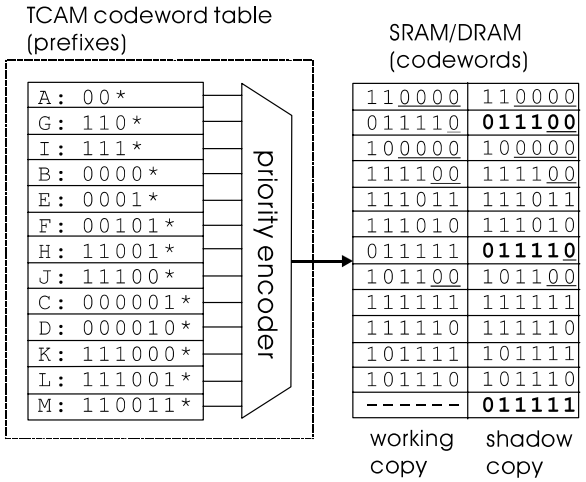


Fig. 6. Updating the codeword table.

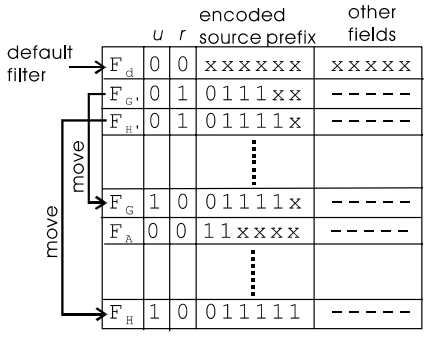


Fig. 7. Maintaining a consistent view of the encoded filter table.

A consistent view of the encoded filter table can be maintained as follows. Two control bits, u (update) and r (replace) are associated with each entry in the encoded filter table as shown in Fig. 7. Normally, bits u and r are set to 0. Two

zero's that correspond to the bit positions of u and r are attached to the input search key. During the update process, the u bit is temporarily masked off in a search operation by programming the global mask register of the TCAM. The system will then set the u bit of the filters that need to be modified. A new copy of the affected filters will be appended to the end of the filter table (before the default filter, if any) in the same relative order in which the original filters appear in the filter table. The r bit of the newly created filter copies will be set to 1. When the system switches over to use the shadow copy of the codeword table, the global mask register of the TCAM is reprogrammed to mask off the r bit and enable the u bit. As a result, the old copies of the affected filters are "disabled" and are replaced by the new copies. The system will then move the new copy of the encoded filters to replace the old copy and reset the u bit and r bit to 0. The insertion of the prefix M and the filter F_M can be carried out after the field values of the existing filters have been updated.

4. Extension to IPv6

Suppose the translation of the source/destination address to codeword is implemented using TCAM. In the straightforward implementation, the TCAM device is configured to have word length of at least 128 bits. A better space saving can be achieved by using a 2-level lookup table organization [12].

We assume the TCAM can be configured to have 72-bit word length (the natural word length of most of the commercial TCAM devices). The prefixes are divided into two groups, $\mathbf{G_S}$ and $\mathbf{G_L}$. $\mathbf{G_S}$ contains all the prefixes with no more than 72 bits, and $\mathbf{G_L}$ contains all the prefixes with more than 72 bits. Commercial TCAM devices can be divided into blocks of 2K 72-bit words, and a search operation can be confined to a selected subset of blocks. The TCAM blocks are divided into two partitions, $\mathbf{P_S}$ and $\mathbf{P_L}$. Prefixes in $\mathbf{G_S}$ are stored in $\mathbf{P_S}$. Prefixes in $\mathbf{G_L}$ are grouped by the value of the first 72 bits. Prefixes a and b are put in the same subgroup if a and b share a common 72-bit prefix $\mathbf{M}$. Assume the number of subgroups of long prefix is less than 64 K. Each subgroup is assigned a distinct 16-bit tag T. The common prefix $\mathbf{M}$ of a subgroup is inserted to the partition $\mathbf{P_S}$ of the TCAM to serve as a *marker*. An entry in the codeword table is a five-tuple $\langle prefix, p, m,$ codeword, $T\rangle$, where the flags p and m indicates whether the entry is a prefix, a marker, or both. For a marker, the codeword field records the codeword of the longest prefix in $\mathbf{G_S}$ that matches $\mathbf{M}$, if any. If the returned codeword is a zero string, then it means that there is not matching prefix found in the codeword table. Let a be a prefix in subgroup $\mathbf{M}$. The bits of a are numbered 1 to l starting from the left. We define the suffix a_s of a as the substring consisting of bits 73 to l. An entry is inserted to partition $\mathbf{P_L}$ of the TCAM by concatenating $\mathbf{a_s}$ to the tag T. The entries in the two partitions of the TCAM are ordered by their length. An example of the 2-level codeword table organization is shown in Fig. 8, where a full-length address has 12 bits and a tag has 4 bits. A wildcard entry is added to each of the two partitions to represent the no match condition.

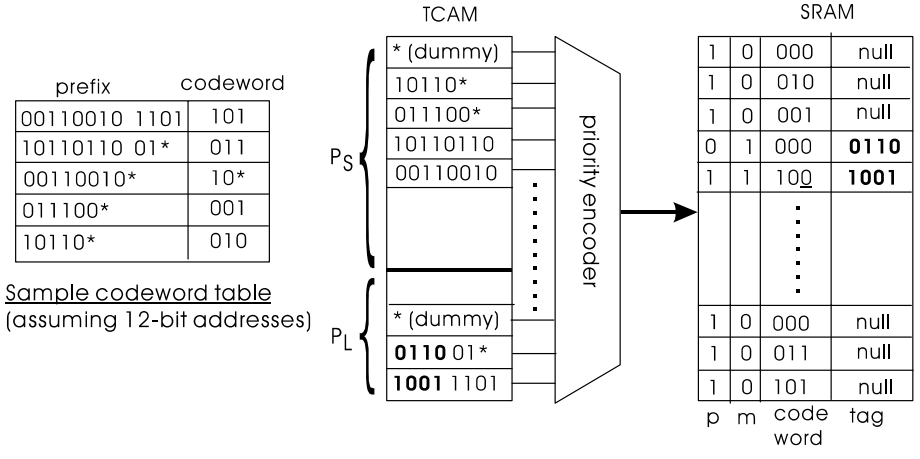


Fig. 8. Organization of a 2-level codeword lookup table.

Codeword lookup is a two-step process. First, the most significant 72 bits of the address **A** are extracted and the partition $\mathbf{P_S}$ is searched. If the best matching entry found is not a marker, the required codeword is obtained. If the best matching entry is a marker, the 56-bit suffix of **A** is extracted and is concatenated with the 16-bit tag of the marker to form a key to search the partition $\mathbf{P_L}$. If the search result is invalid, i.e. a string of zero's is returned, then the codeword value found in step 1 will be used; otherwise the codeword found in step 2 will be used. If there are more than 64 K subgroups of long prefixes, we can further divide $\mathbf{P_L}$ into two or more partitions. In addition to the tag value, a partition ID is associated with the marker. When a partition is searched, the other partitions are disabled. Hence, each partition can support 64 K subgroups of long prefixes, i.e. a tag value only has local context within a partition.

The address prefix in a filter specifies either a network/subnetwork or a specific host. We assume the organization of networks/subnetworks in companies will not be changed substantially when we transit to IPv6. The prefix inclusion properties of IPv6 filter table should be similar to today's IPv4 filter tables. Hence, the expected codeword length for IPv6 filter set will remain more or less the same. In the synthetic IPv4 filter sets, the number of distinct source/destination prefixes is about $N/2$, where N is the number of filters, and half of the prefixes are full-length addresses. In IPv6, the first 64 bits of the address constitute the network/subnetwork ID, and the remaining 64 bits constitute the host/interface ID. Assuming similar prefix length distributions in IPv6 classifiers where 50% of the prefixes are no longer than 72 bits, the amount of TCAM storage required for codeword translation is about $72 \times 1.5 \times N/2$ bits using the 2-level table organization, i.e. about 54 bits per filter. The average expansion factor (due to arbitrary port ranges) for ACL/IPC types of classifiers is about 1.6, and the average expansion factor for firewall type classifiers is about 4. Even if the encoded filters are stored in 72-bit TCAM words (which will allow us to have over 30 spare bits in the code space to handle incremental updates), the TCAM storage saving for ACL/IPC types of classifiers is about 73%, and 90% for firewall type classifiers. If the overhead due to the use of 576-bit word to store a 296-bit IPv6 filter is taken into account, the space saving of PIC is about 84% and 95% for ACL/IPC and firewall type classifiers, respectively.

5. Conclusion

A novel *prefix inclusion coding* (PIC) method to improve the space efficiency of TCAM-based packet classification engines is presented. Comparing with the conventional approach, PIC can reduce the TCAM storage requirement by 70% to over 90%. The reduction of TCAM storage requirement will also alleviate the high power dissipation problem. Unlike P^2C, codeword translation and classification in PIC can be done using TCAM without the needs of other type of special purpose hardware. This helps to lower the overall system cost, especially for medium size classifiers with a few thousands filters. A router may need to maintain more than one filter database. A pure TCAM-based lookup engine allows the sharing of memory resources among the filter databases, e.g. a 18 Mb TCAM device can support multiple medium size lookup tables.

PIC is a very effective encoding method with $O(N \log N)$ computation complexity. We generate synthetic rule sets that are 5 times the size of the original classifiers and find that the overall code length is only increased by 3–5 bits, i.e. the code length of individual fields are increased by 1–2 bits. We have demonstrated that incremental updating can be done efficiently if extra code spaces are made available. When the size of the filter table continues to grow, it may be necessary to rebuild the encoded filter table. The codewords are recomputed and a new copy of the filter table is constructed. The ratio of the TCAM space for the encoded filter table to the TCAM space for the codeword table is about 2 to 1. Suppose the system needs to maintain 3 filter tables, and the overall amount of TCAM space required by the PIC method is 25% of the conventional method. Hence, if the system provides spare TCAM storage for one encoded filter table, the overall space reduction ratio will only be lowered from 75% to 70% when the spare storage spaces are taken into account.

The PIC method can be applied to IPv6 with even better space efficiency, and the lookup rate can be improved. In the conventional approach, a 296-bit IPv6 packet header is entered into the TCAM in 5 iterations via the 72-bit I/O port. By encoding the filters in no more than 72 bits, the search key can be entered in a single step. If separate TCAMs are used for codeword translation and classification, the packet classification rate can be doubled by pipelining the codeword translation and classification steps.

References

[1] I. Arsovski, T. Chandler, A. Sheikholeslami, A ternary content-addressable memory (TCAM) based on 4T static storage and including a current-race sensing scheme, IEEE J. Solid-State Circ. 38 (1) (2003) 155–158.

[2] F. Baboescu, G. Varghese, Scalable packet classification, in: ACM SIGCOMM, 2001.

[3] F. Baboescu, S. Singh, G. Varghese, Packet classification for core routers: is there an alternative to CAMs? in: IEEE INFOCOM, 2003.

[4] H.J. Chao, Next generation routers, Proc. IEEE 90 (9) (2002) 1518–1558.

[5] Cypress Semiconductors Co. Available from: <www.cypress.com>.

[6] P. Gupta, N. McKeown, Packet classification on multiple fields, in: ACM SIGCOMM, 1999, pp. 147–160.

[7] P. Gupta, N. McKeown, Classification using hierarchical intelligent cuttings, IEEE Micro (2001) 34–41.

[8] P. Gupta, N. McKeown, Algorithms for packet classification, IEEE Network (2001) 24–32.

[9] P. Gupta, K. Etzel, J. Bolaria, A scalable and cost-optimized search subsystem for IPv4 and IPv6, EE Times NetSeminar, 28 June 2004. Available from: <http://www.eetimes.com/netseminar.html>.

[10] H. Liu, Efficient mapping of range classifier into ternary-CAM, in: IEEE Symp. High Performance Interconnects (Hotl'02), 2002.

[11] Music Semiconductors. Available from: <www.music-ic.com>.

[12] D. Pao, TCAM organization for IPv6 address lookup, in: IEEE Int. Conf. Adv. Communication Technology, February 2005.

[13] D. Pao, C. Liu, A. Wu, L. Yeung, K.S. Chan, Efficient hardware architecture for fast IP address lookup, IEE Proc.—Comput. Digital Techn. 150 (1) (2003) 43–52.

[14] A. Prakash, R. Kotla, T. Mandal, A. Aziz, A high-performance architecture and BDD-based synthesis methodology for packet classification, IEEE Trans. Computer-Aid. Des. Integrat. Circ. Syst. 22 (6) (2003) 698–709.

[15] R. Sangireddy, A.K. Somani, High-speed IP routing with binary decision diagrams based hardware address lookup engine, IEEE JSAC 21 (4) (2003) 513–521.

[16] S. Singh, F. Baboescu, G. Varghese, J. Wang, Packet classification using multidimensional cutting, in: ACM SIGCOMM'03, 2003, pp. 213–224.

[17] E. Spitznagel, D. Taylor, J. Turner, Packet classification using extended TCAMs, IEEE ICNP (2003).

[18] D.E. Taylor, J.S. Turner, ClassBench: a packet classification benchmark, in: IEEE INFOCOM, 2005. Available from: <http://www.arl.wustl.edu/~det3>.

[19] D.E. Taylor, J.S. Turner, Scalable packet classification using distributed crossproducting of field labels, in: IEEE INFOCOM, 2005.

[20] J. van Lunteren, Searching very large routing tables in wide embedded memory, in: IEEE GLOBECOM 2001, pp. 1615–1619.

[21] J. van Lunteren, T. Engbersen, Fast and scalable packet classification, IEEE JSAC 21 (4) (2003) 560–571.

[22] Z. Wang, H. Che, M. Kumar, S.K. Das, CoPTUA: Consistent policy table update algorithm for TCAM without locking, IEEE Trans. Comput. 53 (12) (2004) 1602–1604.

Derek Pao received the Ph.D. in Computer Science from Concordia University, Montreal, Canada, in 1991. He is an Associate Professor of Electronic Engineering at the City University of Hong Kong. His research interests include IP address lookup and packet classification algorithms, network intrusion detection, communications protocols, and wireless ad hoc networks.

Yiu Keung Li received the B.Sc. with 1st class honours and M.Phil. degrees in Information Technology from the City University of Hong Kong in 2002 and 2005, respectively. His research interests include IP address lookup and packet classification algorithms, high-speed data network, quality of service in IP network, distributed computing and software engineering.

Peng Zhou was born in Shanghai, China. He was initially an undergraduate student of the Shanghai Jiao Tong University, China. He was awarded the Hong Kong Jockey Club Scholarship in 2002 for outstanding Mainland students to study in Hong Kong, and received the B.Eng. in Computer Engineering with 1st class honours from the City University of Hong Kong in 2005. He is currently pursuing postgraduate studies at the same university.