Scalable packet classification by TCAM entry encryption algorithm

Chun-Liang Lee and Pi-Chung Wang

Department of Computer Science and Information Engineering, Chang Gung University, Taoyuan 333, Taiwan
E-mail: cllee@mail.cgu.edu.tw

Institute of Networking and Multimedia, National Chung Hsing University, Taichung 402, Taiwan
E-mail: pcwang@cs.nchu.edu.tw

Department of Computer Science and Engineering, National Chung Hsing University, Taichung 402, Taiwan
E-mail: pcwang@cs.nchu.edu.tw

Abstract. Ternary content-addressable memory (TCAM) has been widely used to perform fast packet classification due to its ability of solving the problem in $O(1)$ time without considering the number of entries, mask continuity and their lengths. However, it also comes with several shortcomings, such as the limited number of entries, expensive cost and power consumption. We propose an efficient algorithm to reduce the required TCAM bits by encoding the IP address portion of the policies. With the proposed scheme, the achieved compression ratio can be as small as 9% for IPv6 policies. Moreover, the TCAM throughput is improved by a factor of 175% with pipelining design. Obviously, the proposed scheme is attractive for the forthcoming IPv6.

Keywords: Internet, high-speed network, packet classification, IPv6, ternary content addressable memory

1. Introduction

Packet classification is a problem of multi-dimensional range match. Each classifier maintains a set of policies to categorize an incoming packet stream into multiple classes. A policy $F = (f[1], f[2], \ldots, f[k])$ is called $k$-dimension if the policy consists of $k$ fields, where each $f[i]$ is either a variable length prefix bit string, a range or an explicit value of the packet header. A policy can be any combination of fields of the packet header. The most common fields are the IP source address, the IP destination address, the protocol type and port numbers of source/destination applications. A packet $P$ is said to match a particular policy $F$ if for all $i$, the $i$th field of the header satisfies $f[i]$. Each policy has an associative action, which is used to process the related packet.

TCAM is one popular hardware device to perform fast packet classification. As compared to the software-based solutions, the TCAM can offer sustained throughput and simple system architecture, thus makes it attractive. However, it also comes with several shortcomings, such as size limitation, power consumption and expensive cost. In this letter, we propose an efficient algorithm to reduce the required TCAM by encoding the IP prefix portion of policies. The new scheme could reduce the length of TCAM entries from $W$ to $(\log_2 N + \beta)$, where $W$ is the length of IP address, $N$ is the number of the distinct prefixes in policy database and $\beta$ is the maximal number of sub-prefixes for each prefix. According to the related work [1,2], $\beta$ is usually small as compared with the maximal prefix length. In our experiments, the proposed scheme could encrypt the 128-bit prefixes of the real-world IPv6 routing tables into 11 bits. With the proposed scheme, the achieved compression ratios for policy databases with 1K to 1M entries vary from 34–66% for IPv4 policies and 8–16% for IPv6 policies, respectively. Furthermore, the TCAM throughput can be improved by a factor of 175% with pipelining design.

*Corresponding author.
The rest of the paper is organized as follows. Section 2 presents related work. The proposed algorithm is addressed in Section 3. The experiment results are presented in Section 4. Finally, a summary is given in Section 5.

2. Related work

Several algorithms for classifying packets have recently appeared in the literature [3–10]. They can be grouped into the following classes: linear search/caching, hardware-based solutions, grid of tries, hash-based solutions, decision-based schemes, and combination-based schemes. The following briefly describes the important properties of these algorithms. Assume that \( N \) is the number of the filters, \( k \) is the number of classified fields and \( W \) is the length of the IP address.

Linear search/caching: The simplest method for packet classification involves a linear search of all the filters. The spatial and temporal complexity is \( O(N) \). Caching is a technique frequently used at either the hardware or the software level to improve the performance of linear search. However, the performance of the caching depends critically on whether if each flow is of a large packet size. Also, if the number of simultaneous flows exceeds the cache size, then the performance would be severely degraded.

Hardware-based solutions: A high degree of parallelism can be implemented in hardware to provide a speed-up advantage. In particular, ternary content addressable memories (TCAMs) can be used effectively to look up filters. However, TCAMs with a particular word width cannot be used when flexible filter specification is required. Manufacturing TCAMs with sufficiently wide words to contain all bits in a filter is difficult. It also suffers from the problem of power consumption and scalability [11]. Lakshamn et al. presented another scheme that depends on a very wide memory bus [6]. The algorithm reads \( Nk \) bits from memory, corresponding to the best matching prefixes (BMPs) in each field, and determines their intersection to find a set of matching filters. The memory requirement for this scheme is \( O(N^2) \). The hardware-oriented schemes rely on heavy parallelism and involve a considerably high hardware cost. In addition, the flexibility and scalability of hardware solutions are very limited.

Grid of tries: Specifically for the case of two-field filters, Srinivasan et al. [5] presented a trie-based algorithm. The algorithm has a memory requirement of \( O(NW) \) and requires \( 2W - 1 \) memory accesses per filter lookup. An enhanced version is presented in [12].

Hash-based solution: This solution is motivated by the observation that, albeit filter databases include several different prefixes or ranges, the distinct prefix lengths tend to be rare [4]. For example, backbone routers have around 60K destination address prefixes, but actually only 32 distinct prefix lengths exist. Hence, all the prefixes can be divided into 32 groups, one for each length \( (W) \). Since all prefixes in a group have the same length, the prefix bit string can be used as a hash key, leading to a simple IP lookup scheme, which requires \( O(W) \) hash lookups, independent of the number of prefixes. The algorithm of Waldvogel [13] performs a binary search over the \( W \) length groups, and has a worst-case time complexity \( O(\log W) \). The tuple space idea [4] has given rise to two-dimensional filters in a foregoing approach [13]. A tuple is a set of filters with specific prefix lengths, and a set of tuples is called a “tuple space”. Since each tuple has a specific bit-length for each field, these bit-lengths can be concatenated to create a hash key, which can be used in performing the tuple lookup. The matched filter can be found by probing each tuple alternately, and tracking the least-cost filter. For example, the two-dimensional filters \( F = (10*, 110*) \) and \( G = (11*, 001*) \) both belong to the tuple \( T_{2,3} \) in the second row and third column in the tuple space. When searching for \( T_{2,3} \), a hash key is constructed by concatenating two bits of the source field with three bits of the destination field. Even a linear search in the tuple space represents a considerable improvement over a linear search of the filters since the number of tuples is typically much smaller than the number of filters. Rectangle search and
pruned tuple space search are designed to improve the performance of tuple space search. In [14,15], the speed and storage performance of the rectangle search are improved by reducing the number of tuples.

**Decision-based solution:** Work on decision-based algorithms includes papers presented by Gupta et al. [10] and Woo [9]. Both schemes use a decision tree to divide the filters into multiple groups. Each group is listed in the leaf nodes of the decision tree, and linear search is used to traverse the group. The number of filters in each group is limited by a pre-defined value. The decision at each node could be a field [10] or a bit of any field [9]. A suitable selection of decisions would minimize the required storage and search time. The hypercuts presented by Singh et al. [16] further extends the single-dimensional decision into a multi-dimensional one.

**Combination-based solution:** A general mechanism, called cross-producting, involves BMP lookups on individual fields and the use of a pre-computed table to combine the results of individual prefix lookups [5]. However, this scheme suffers from a $O(N^k)$ memory blowup for $k$-field filters. Gupta et al. presented an algorithm that can be considered to be a generalization of cross-producting [3]. After BMP lookup is performed, a recursive flow classification algorithm hierarchically performs cross-producting. Thus $k$ BMP lookups and $k-1$ additional memory accesses are required per filter lookup. The algorithm is expected to improve the average throughput significantly; nevertheless, it requires $O(N^k)$ memory in the worst case. Also, in the case of two-field filters, this scheme is identical to cross-producting and, hence, has a memory requirement of $O(N^2)$.

3. **TCAM entry encryption algorithm**

In [1], Baboescu and Varghese revealed that no prefix contains more than 4 matching sub-prefixes in each dimension of the ISP/firewall policy databases. The characteristic conforms to the nature of network-layer address/mask specification described in [17]. This is mainly because that the service boundaries specified by the policies are highly related to those specified by the network-layer prefixes, such as ISP VPN policies. In addition, the identical IP prefixes are usually used in various policies. We are inspired to encode the prefixes used in the policy database as much shorter ones so that the required TCAM is reduced. To achieve the purpose, we have to realize the nature of IP routing prefixes first.

The adoption of classless inter-domain routing (CIDR) [18] allows the network administrator to specify a smaller network within an existing network. For example, an ISP network is specified by prefix $\langle 206.95.\ast \rangle$. It might exist an enterprise customer network, which is specified by prefix $\langle 206.95.130.\ast \rangle$. The encoding scheme must be able...
to keep the hierarchical properties of the routing prefixes. Namely, the encrypted key of \((206.95.\ast)\) must be a sub-prefix of that of \((206.95.130.\ast)\).

To encode the address portion of policies, a straightforward scheme is to divide the prefixes into several bit-streams according to the length of their shorter prefixes, as shown in Fig. 1. The prefix \(P_2\) \((010000)\) has two shorter prefixes: \(P_1\) \((0)\) and \(P_3\) \((0100)\). Thus it is divided into three bit-streams \((0), (100), (00)\) that are inserted to bit-stream group “Level 1”, “Level 2” and “Level 3”, respectively. It might derive the same bit-streams from different prefixes; for example, the “Level 2” bit-stream \((01)\) of prefix \(P_{13}\) is identical to that of \(P_2\). In each group, the duplicate bit-streams are eliminated and each bit-stream is assigned an unique ID. By concatenating the relevant IDs, the encrypted key for each prefix is generated. In this example, there are 3 bit-streams in “Level 1”, 5 in “Level 2” and 4 in “Level 3”. Thus maximal \(7 \,(= \,2 + 3 + 2)\) bits are required to represent the original prefixes.

Though the basic scheme is simple, the encoding results are inefficient since the exclusive relation between prefixes is not utilized. For example, the “Level 2” bit-streams \((01), (100)\) and \((1010)\) \((i.e., \,corresponding \,to \,P_2, \,P_3 \,and \,P_7)\) only concatenate to \((0)\) \((i.e., \,P_1)\). Based on the number of bit-streams attached to a specific sub-prefix, there are three bit-streams connected to \((0)\) \((P_1)\) and two to both \((101)\) \((P_6)\) and \((11)\) \((P_{12})\). Therefore, the number of bits for “Level 2” could be reduced to two and the total length is reduced to six.

With ingenious enhancements, we can further improve the bit-stream encoding called adaptive scheme. According to the successive bit-stream length, the encoder adjusts the length of encoded ID dynamically. It encodes the bit-stream with bottom-up manner and uses “Huffman Encoding” to reduce the maximal length of the concatenated IDs. As shown in Fig. 2, the bit-streams in “Level 3” are encoded at first, and the ID length for each bit-stream is recorded. The “Level 2” bit-streams are sorted according to the ID length of their successive bit-streams. Then the longest one, for example, the “Level 2” bit-streams \((1)\) corresponding to \(P_8\) is assigned the shortest ID “0”. Another “Level 2” bit-stream \((0)\) attached to \(P_6\) is thus assigned an exclusive ID “1”. The bit-streams with the shortest IDs are marked with black circles. The IDs for each bit-stream are listed in the left part of Fig. 2. In Table 1, we show the encrypted prefixes for different schemes. With the basic scheme, the maximal required length is 7. It can be improved to 4 with the adaptive scheme.

The pseudo code of the prefix encoding algorithm is presented in below. The main portion of the algorithm \textit{“Encoder”} is called recursively. “Encoder” will traverse the binary tree of the prefixes in depth-first order. If the currently traversed node corresponds to a prefix, it encodes the bit-streams according its successive ID length.
Table 1
The encrypted prefixes for different schemes

<table>
<thead>
<tr>
<th>Basic</th>
<th>Adaptive</th>
<th>Basic</th>
<th>Adaptive</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>00 10</td>
<td>$P_8$</td>
<td>01 10 00</td>
</tr>
<tr>
<td>$P_2$</td>
<td>00000 1010</td>
<td>$P_9$</td>
<td>01 01 00 00</td>
</tr>
<tr>
<td>$P_3$</td>
<td>0001 100</td>
<td>$P_{10}$</td>
<td>0110 0001</td>
</tr>
<tr>
<td>$P_4$</td>
<td>00010 1000</td>
<td>$P_{11}$</td>
<td>1011 0010</td>
</tr>
<tr>
<td>$P_5$</td>
<td>00010 1011</td>
<td>$P_{12}$</td>
<td>10 11</td>
</tr>
<tr>
<td>$P_6$</td>
<td>000010 1000</td>
<td>$P_{13}$</td>
<td>1000 110</td>
</tr>
<tr>
<td>$P_7$</td>
<td>0011 01</td>
<td>$P_{14}$</td>
<td>10100 111</td>
</tr>
</tbody>
</table>

by executing “Huffman-Encoding” and records the IDs. Since each node is traversed exactly once, the time complexity is $O(NW)$. The encoded prefixes are derived by concatenating IDs, which are fetched along the path from the root to the prefix node.

Prefix Encoding Algorithm

**Input:** The root of the binary tree constructed from the prefixes referred in policy database

**Output:** The IDs of the prefixes

Encoder(Current, Parent, Level) BEGIN

IF (Current→Prefix = true) BEGIN

M=Encoder(Current→Left, Current, Level+1);
N=Encoder(Current→Right, Current, Level+1);
L=Huffman_Encoding(Current→List, M+N);
Current→EncodingLength=L;
Add_List(Parent, Current);
return 1;
END
ELSE BEGIN

M=Encoder(Current→Left, Parent, Level+1);
N=Encoder(Current→Right, Parent, Level+1);
return M+N;
END
END

/* Huffman_Encoding(List, Count) encodes the attached bit-streams according to the length of the attached IDs. */

/* Add_List(Parent, Current) appends the corresponding prefix in the current node to its parent prefix node for encoding later. */

/* The encoded prefixes are derived by concatenating IDs, which are fetched along the path from the root to the prefix node in the binary tree. */

In each classifier, the used prefixes are extracted from the policies to constructs the prefix tree. Then the proposed scheme encrypts the prefixes and replaces the address portion of the policies by the IDs. The encrypted source/destination prefixes and the policies are stored in separated TCAMs, as shown in Fig. 3.

To accomplish packet classification, two TCAM accesses of source and destination addresses are performed to find out the IDs corresponding to the best matching prefixes. The address prefixes are replaced by the IDs and forwarded to the classifier. Consequently, the classifier performs packet classification by the policy TCAM to derive the service priority. The TCAM accesses can be pipelined to increase the throughput. Moreover, each TCAM access can be fastened by benefiting from fewer READ cycles since the lengths of the searched keys are...
shortened. To evaluate the packet classification, the TCAM provided by the Network Search Engine (NSE) [19] is used to examine the performance. The whole classification operations include three operations: read data, lookup and output the result. The data bus is 72 bits, and the input of the five fields of an IPv6 header (296 bits) takes 5 clock cycles. Then the NSE only takes one clock to lookup and another clock cycle to output the result from SRAM. With the pipelined implementation, the total clock cycles can be eliminated to 4. The latency comparisons of original (without TCAM compression) and adaptive schemes are shown in Table 2. For the adaptive scheme, the values in front of the slashes are the required clock cycles without pipelining and those behind the slashes are the required cycles with pipelining. Although the adaptive scheme maintains the latencies just the same as that of original scheme in IPv4, it improves the speed by a factor of 175% with pipelining in IPv6.

4. Performance evaluation

Through experiments, we demonstrate that the proposed scheme features much less TCAM bits. Currently, the IPv6 routing tables consist of only few hundreds prefixes which are download from 6bone [20]. We also use the real data available from the IPMA [21] and NLANR [22] projects for comparison, these data provide a daily snapshot of the routing tables used by some major Network Access Points (NAPs). The maximal lengths of the encoded prefixes for different routing tables and different schemes are illustrated in the results.

Table 3 shows the encoding results with different routing tables. For the IPv4 routing tables, the basic scheme incurs longer encoded-bits than the original prefixes (32 bits). It is because the scheme concatenates maximal bits
C.-L. Lee and P.-C. Wang / Scalable packet classification by TCAM entry encryption algorithm

Table 3
The maximal length of encoded prefixes for different IPv4 routing tables

<table>
<thead>
<tr>
<th>Site</th>
<th>Number of prefixes</th>
<th>Basic scheme</th>
<th>Adaptive scheme</th>
<th>Minimal length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paix</td>
<td>13,395</td>
<td>33</td>
<td>16</td>
<td>13.71</td>
</tr>
<tr>
<td>PacBell</td>
<td>24,770</td>
<td>40</td>
<td>19</td>
<td>14.60</td>
</tr>
<tr>
<td>AADS</td>
<td>32,388</td>
<td>41</td>
<td>19</td>
<td>14.98</td>
</tr>
<tr>
<td>Mae-West</td>
<td>36,943</td>
<td>40</td>
<td>19</td>
<td>15.17</td>
</tr>
<tr>
<td>Mae-East</td>
<td>58,101</td>
<td>47</td>
<td>19</td>
<td>15.83</td>
</tr>
<tr>
<td>NLANR</td>
<td>102,271</td>
<td>64</td>
<td>21</td>
<td>16.64</td>
</tr>
</tbody>
</table>

Table 4
The maximal length of encoded prefixes for different IPv6 routing tables

<table>
<thead>
<tr>
<th>Number of prefixes</th>
<th>Basic scheme</th>
<th>Adaptive scheme</th>
<th>Minimal length</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>6</td>
<td>6</td>
<td>5.64</td>
</tr>
<tr>
<td>142</td>
<td>11</td>
<td>8</td>
<td>7.15</td>
</tr>
<tr>
<td>284</td>
<td>14</td>
<td>9</td>
<td>8.15</td>
</tr>
<tr>
<td>495</td>
<td>19</td>
<td>11</td>
<td>8.95</td>
</tr>
</tbody>
</table>

Table 5
The compression ratios of the required TCAM bits

<table>
<thead>
<tr>
<th>Policies (x)</th>
<th>N = x/10</th>
<th>N = x/20</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv4</td>
<td>2^10</td>
<td>2^15</td>
</tr>
<tr>
<td></td>
<td>35%</td>
<td>50%</td>
</tr>
<tr>
<td>IPv6</td>
<td>9%</td>
<td>13%</td>
</tr>
</tbody>
</table>

for each level to generate the encrypted prefixes. Contrarily, the adaptive scheme could complement the longest ID with shortest one to eliminate the total length. In the experimental results, the adaptive scheme could encrypt the 32-bit prefix to a 22-bit one, which shows a bit-reduction of 70%. The encrypted ID length is larger than the optimal encoding value \( \log_2(\text{number of prefixes}) \). This is caused by integer rounding-up at each level. For example, in the routing table of NLANR, there are 102,271 prefixes and 6 levels. The maximal length for NLANR table is 22-bit which is close to \( \log_2(102,271) + 6 = 22.6 \).

For the IPv6 routing tables, the adaptive scheme still outperforms the basic one, as shown in Table 4. Since the numbers of prefixes and the maximal number of sub-prefixes for each prefix are less than that in IPv4, even the simplest scheme could achieve significant reduction. While the number of prefixes increases, the effect of the adaptive scheme is appeared. The adaptive scheme could encrypt the prefix from 128 bits to 11 bits, which shows a bit-reduction of 9%.

Through experiments, the required bits are constrained from the increasing prefix length and prefix count. As the routing table contains 1M entries and 12 sub-prefixes, it will require about 32 bits for encryption, not to mention the route aggregation in IPv6 would largely reduce the number of sub-prefixes.

The required bits for the prefixes in the policy databases can be expressed as \( |F| \times (\lceil \log_2 N \rceil + \beta) + N \times W \), where \( |F| \) is the number of policies, \( |F| \times (\lceil \log_2 N \rceil + \beta) \) is the number of the required bits for the policies, and \( (N \times W) \) is that for the distinct prefixes. According to the previous work [7], each source prefix is usually used on average between 19 to 35 times and the destination prefix is used on average between 11 to 14 times in policy databases. Assuming that each prefix is used on average 10 and 20 times, and \( x \) is the number of policies. As shown in Table 5, the compress ratio is reduced as the number of policies increases. This is mainly because the number of distinct prefixes is increased as well as the number of policies, thus the required encrypted bits are also raised.
The efficiency for IPv6 is better than that for IPv4 since the performance of the proposed scheme ties to only the number of distinct prefixes and the maximal number of sub-prefixes, rather than the prefix length. Hence, the IPv6 packet classification can benefit from the proposed scheme significantly.

5. Conclusions

This study investigates the major issues in TCAM-based classifier design. To make use of the TCAM in IPv6 packet classification, we propose an approach to utilize the limited TCAM storage. By encoding the prefixes into much shorter ones, the required bits for the TCAM entry could be significantly reduced. The fundamental idea is to divide the prefixes according to the length of their shorter prefixes and allocate enough bits for each level. The scheme could be further improved by adopting the concept of exclusion to eliminate the combination in each level and using Huffman encoding. In our experiments, a typical IPv4 routing prefix needs 22 bits and 11 bits for IPv6 routing prefix in the worst case. With the proposed scheme, the compression ratios of the required storage for policy databases with 1K to 1M entries vary from 31–66% for IPv4 prefixes and 8–16% for IPv6 prefixes, respectively. With the pipelining design, the proposed scheme can outperform the original design by a factor of 175%. Obviously, the proposed scheme is attractive for the forthcoming IPv6.

Acknowledgements

This work is supported in part by the National Science Council under Grant Nos NSC 95-2221-E-182-073 and NSC 96-2221-E-005-010.

References

[21] Merit Networks Inc., Internet performance measurement and analysis (IPMA) statistics and daily reports, IPMA Project, see http://www.merit.edu/ipma/routing_table/.
[22] NLANR Project, National Laboratory for Applied Network Research, see http://www.nlanr.net.