Available online at www.sciencedirect.com

ScienceDirect

Computer Communications 29 (2006) 3037–3050

computer communications

www.elsevier.com/locate/comcom

A scalable IPv6 route lookup scheme via dynamic variable-stride bitmap compression and path compression ☆

Kai Zheng *, Zhen Liu, Bin Liu

Department of Computer Science, Tsinghua University, Beijing 100084, PR China

Available online 10 January 2006

Abstract

The significantly increased address length of IPv6 (128-bit) provides an endless pool of address space. However, it also poses a great challenge on wire-speed route lookup for high-end routing devices, because of the explosive growth of both lookup latency and storage requirement. As a result, even today's most efficient IPv4 route lookup schemes can hardly be competent for IPv6. In this paper, we develop a novel IPv6 lookup scheme based on a thorough study of the distributions of real-world route prefixes and associative RFC documents. The proposed scheme combines the bitmap compression with path compression, and employs a variable-stride mechanism to maximize the compress ratio and minimize average memory reference. A possible implementation using mixed CAM devices is also suggested to further reduce the memory consumption and lookup steps.

The experimental results show that for an IPv6 route table containing over 130K prefixes, our scheme can perform 22 million lookups per second even in the worst case with only 440 Kbytes SRAM and no more than 3 Kbytes TCAM. This means that it can support 10 Gbps wire-speed forwarding for back-to-back 40-byte packets using on-chip memories or caches. What's more, incremental updates and high scalability is also achieved.
© 2005 Elsevier B.V. All rights reserved.

Keywords: CAM; Compression; IPv6; Route lookup

1. Introduction

Internet Protocol Version 6 (IPv6) is one of the key supporting technologies of the *Next Generation Network* (NGN). The most distinctive feature of IPv6 is its 128-bit address, which is significantly increased from IPv4 and provides an extremely large pool of address space for the Internet. However, this innovation also poses a great challenge on the data path functions of IP packet forwarding, such as IP address lookup. This pressure is caused mainly by the following two factors: (1) the performances of most existing IP lookup algorithms, including lookup throughput and storage requirement, are sensitive to the key length and will decrease dramatically when migrated to IPv6. (2) The rapid advances in fiber-optic technology push the link speed of backbone from 2.5 to 10 Gbps or even 40 Gbps, making wire-speed forwarding much more difficult to guarantee. Therefore, an effective *Longest Prefix Matching* (LPM) algorithm for 128-bit IP address is very essential for routers/switches deployed at future networks.

Currently, packet forwarding based on CIDR IPv4 address lookup is well understood with both trie-based algorithms and TCAM-based mechanisms in the literature. trie-based algorithms are usually said to be time consuming, and commonly a lot of memory accesses are needed for a single address lookup. Moreover, most of them [1–4] can hardly be scaled to support IPv6, because their lookup time grows linearly with the search key (Destination IP Address) length. Some algorithms have constant search

☆ This work was supported by NSFC (Nos. 60173009 and 60373007), China 863 High-tech Plan (Nos. 2002AA103011-1 and 2003AA115110), China/Ireland Science and Technology Collaboration Research Fund (CI-2003-02), and the Specialized Research Fund for the Doctoral Program of Higher Education of China (No. 20040003048).

* Corresponding author. Tel.: +8610 6277 3441; fax: +8610 6277 3616.
E-mail addresses: Zk01@mails.tsinghua.edu.cn (K. Zheng), Liuzhen02@mails.tsinghua.edu.cn (Z. Liu), Liub@tsinghua.edu.cn (B. Liu).

0140-3664/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2005.11.003

time for IPv4, such as DIR-21-3-8 [5]. However, their high storage requirements only allow them to be implemented with mass-but-slow SDRAMs. What's more, the exponentially increased memory requirements with the search key length make them not suitable for IPv6 implementation.

The Lulea [6] algorithm, for the first time, develops the concept of bitmap compression to improve storage efficiency, achieving both small memory requirement and almost constant lookup time. Hence much faster SRAM chips can be employed to replace SDRAMs. However, its specially designed memory organization is un-scalable for IPv6 or large route tables. Furthermore, it is known to be very hard to update since it needs leaf-pushing. The Eatherton's Tree Bitmap algorithm [7] improves upon Lulea by creating a data structure (with two kinds of bitmaps, Internal Bitmap and External Bitmap) which does not require leaf-pushing and therefore supports fast incremental updates. An implementation of the Tree Bitmap algorithm, referred to as Fast IP Lookup (FIPL) [8], shows that a storage requirement of about 6.3 bytes per prefix and performance of over one million lookups per FIPL engine can be achieved. However, FIPL uses a fixed lookup stride of four at each level, hence for each IP address lookup it may require more than $32/4 = 8$ memory accesses in the case of IPv4, and $128/4 = 32$[1] in the case of IPv6, which make it also infeasible for IPv6 migration.

Ternary Content Addressable Memory (TCAM) is a fully associative memory that allows a "don't care" state to be stored in each memory cell in addition to 0s and 1s. Since the contents of a TCAM can be searched in parallel and the first matching result can be returned within only a single memory access, TCAM-based scheme is very promising in terms of building a high-speed LMP lookup engine [9,10]. Moreover, TCAM-based tables are typically much easier to manage and update than trie-based ones. However, on the other hand, the high cost to density ratio and high power consumption of TCAM prevent it from being widely adopted in building route lookup engines. This situation will get even worse when migrated to IPv6, since the storage requirement for TCAM-based tables also grows linearly with the address length.

A good IP (v6) address lookup scheme should have the following features. (1) High enough lookup throughput to support high-speed interface even in the worst case (i.e., minimum-sized packets coming in back-to-back); (2) small memory requirement, making it practical to be implemented with small but fast memory chips or on-chip caches; (3) scalability with the key length, maintaining the lookup time, and memory requirement at a feasible low level when migrated to IPv6; (4) Low update cost, to cope with the instability of the BGP protocol.

In this paper, we develop a novel lookup scheme which can achieve all the features mentioned above. The main contributions of the paper are threefold: first, we introduce a series of trie-associated concepts and terms, presenting a new and explicit view of the IP lookup problem and the related techniques/methods. Second, by analyzing real-world IPv4 route tables and several IPv6 "initial allocation" route tables, we not only provide a number of informative observations on distributions of route prefixes and several useful heuristics for lookup scheme designing, but also build an IPv6 prefix generator according to these observations and the recommendations from associative RFC and RIPE documents. This is very essential for current IPv6 lookup scheme performance evaluations, since the available "future-like" IPv6 route tables are not sufficient. Finally, based on those observations, we develop a novel *Dynamic Variable-Stride Bitmap Compressing lookup scheme with Path Compression* (DVSBC-PC) and a further optimized scheme with mixed *CAM device* (DVSBC-PC-CAM).

The rest of the paper is organized as follows. Section 2 presents several trie-associated concepts and terms. In Section 3, we list a number of informative observations on distributions of route prefixes, present several useful heuristics for lookup scheme designing, and develop an IPv6 prefix generator for performance evaluation. Section 4 describes the dynamic variable-stride bitmap compressing IP lookup scheme with path compression (DVSBC-PC) and a further optimized scheme with mixed CAM mechanism (DVSBC-PC-CAM). Section 5 presents the experimental results and performance evaluation of the proposed scheme, as well as the comparison with other schemes. Finally, a conclusion is drawn in Section 6.

2. Definitions and terms for IP address lookup and the trie system

In our opinion, the prefix trie system is more than a tool to introduce trie-based algorithms. In addition to an important data structure in both route table organization and route lookup implementation, it is also very helpful in acquiring a better understanding of the LPM problem. In this section, for the sake of providing intuitions for IP address lookup scheme design and presenting an explicit view of the trie system, we introduce/develop a set of definitions and terms, which will be used throughout the paper.

2.1. Definition: IP address

The bits in a specific IPv6 address are ordered as shown in Fig. 1, where the 1st bit (MSB) lies in the leftmost position and the 128th bit (LSB) lies in the rightmost position. And we let IP[$i - j$] denotes the set of the ith to jth bits of an IP address.

1	2	3	4	......	127	128

Fig. 1. IP address format.

[1] Actually, as reported by the authors, three additional memory accesses are needed.

2.2. Definition: trie, nodes; prefix depth and prefix level

A *Full Binary Trie* is introduced to represent the whole prefix space, with each node for a possible prefix. The prefix of a route table entry defines a path in the trie ending in some node, which is called the *Prefix Nodes* in this paper. If a node itself is not a prefix node but its descendants include prefix nodes, we call it an *Internal Node*. Node that is either a prefix node or an internal node is called *Genuine Node*, which indeed carries route information. We name a prefix node as a *Prefix Leaf* if none of its descendants is a prefix node. Fig. 2 depicts an example prefix trie and can be used to explain these definitions.

Depth of a prefix (node) is defined as the number of its ancestor nodes in the prefix trie. For instance, in the example of Fig. 2 the depths of prefix node A, D, and F are 0, 4, and 6, respectively. For a given prefix node n, there may be multi paths from n to its multi descendant leaf nodes. Among these paths, let $P_{\max}$ be the one containing the most prefix nodes. The number of prefix nodes (excluding node n itself) in $P_{\max}$ is called the *Prefix Level* of prefix node n. In the example of Fig. 2, the level of prefix node A, D, and F are 2, 0, and 1, respectively.

2.3. Definition: sub-trie, sub-trie hierarchy

The term *Sub-Trie* is defined to be a full binary trie that is carved out from a prefix trie. It can be specified by a 2-tuple (*root*, *height*), where *root* is the root node of the sub-trie, and *height* indicates the size of the sub-trie, as shown in Fig. 3. We call the nodes on the bottom of the sub-trie its *Edge Nodes*, and the sub-trie should have 2^{height} edge nodes. For instance, the height of sub-trie#3 in Fig. 3 is 2, and so it has $2^2 = 4$ edge nodes, two of which are genuine nodes (one internal node and one prefix node).

A prefix trie can be partitioned into a set of sub-tries in different ways. If the partitioned sub-tries set satisfy: (1) it covers all of the genuine nodes on the prefix trie; (2) the root nodes of any sub-tries are the edge nodes of other sub-tries, we call it a *sub-trie hierarchy* of the prefix trie. Fig. 3 also shows a possible sub-trie hierarchy. Note that sub-trie hierarchy is also not unique for a given prefix trie.

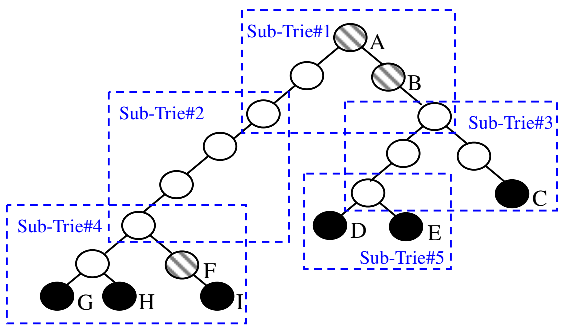


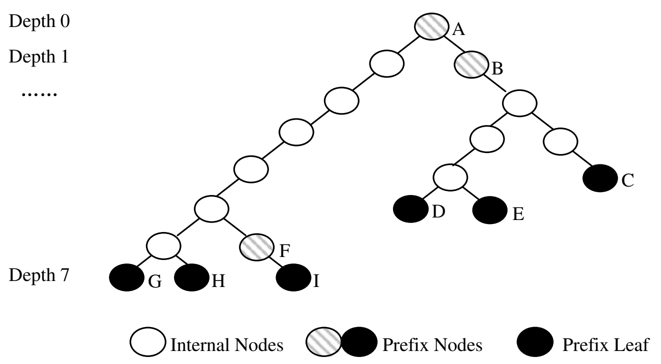
Fig. 3. A possible sub-trie hierarchy of the prefix trie introduced in Fig. 2. Again, we omit the non-genuine nodes in this figure. We see that the prefix trie is partitioned into 5 sub-tries, and they form a sub-trie hierarchy with 3 levels.

As will be explained in the later sections, different prefix trie partitioning may lead to different performance for certain algorithms (e.g., the bitmap compression based algorithms). A proper way of partitioning is, therefore, very important for that kind of algorithms.

2.4. Definition: Pointer Array, Compressed Pointer Array

Each genuine node carries either Next hop IP address (NIP) or trie structure information. We use a data structure called *Pointer Array* (PA) to hold the route information carried by the edge nodes of a sub-trie, with each pointer representing the memory location of either the NIP or the next level sub-trie structure, as shown in Fig. 4.

As depicted in Fig. 4, two successive pointers (for the second and third edge nodes, respectively) within in the PA contain the same value (NIP2). In order to reduce the storage requirement, we can keep only one such information in a *Compressed Pointer Array* (CPA) by introducing a bitmap to indicate which values are omitted. As is illustrated in lower part of Fig. 4, NIP2 appears only one time in the Compressed Pointer Array. The ‘0’ in the bitmap shows that the corresponding pointer in the original PA is omitted, and IP address matching the prefix corresponding to the 3rd edge node should follow the $3 - 1 = 2$nd (since there is only one ‘0’s in the first 3 bits



Route Table

No	Prefix	Next Hop
A	*	NIP1
B	1*	NIP2
C	1111*	NIP3
D	11000*	NIP3
E	11001*	NIP1
F	000001*	NIP4
G	0000000	NIP2
H	0000001	NIP1
I	0000011	NIP1

Fig. 2. An example routing table and the corresponding binary trie built from it. For clarity and simplicity, we omit the trivial nodes (non-genuine nodes) in the figure.

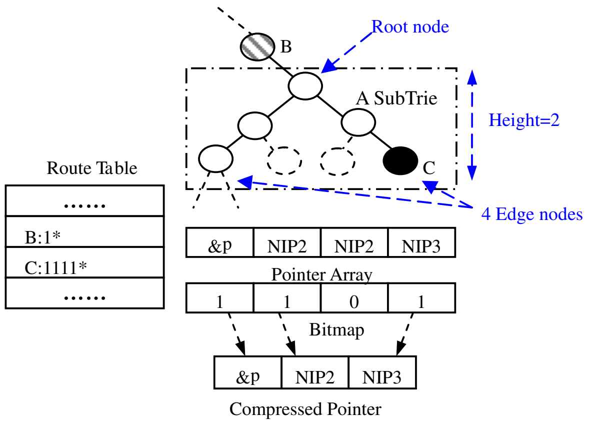


Fig. 4. PA (Pointer Array) and the corresponding Compressed PA. This figure depicts part of the prefix trie introduced in Fig. 2. The sub-trie has 4 edge nodes and they are associated with a PA with 4 pointers, each of which contains the memory location of either the NIP or the next sub-trie structure.

of the bitmap, we know that only one pointer has been omitted in the PA) pointer in the CPA, i.e., NIP2.

Such idea/technique is called *Bitmap Compression*, which has been adopted in several algorithms. Although it is hard to scale to IPv6 in its original form, it is useful in providing intuitions for memory-saving lookup scheme.

CPA Density (CPAD) of a sub-trie is defined as the ratio of the number of pointers in the CPA to the total number of edge nodes of the sub-trie. This parameter indicates the compressing ratio/potential of a sub-trie structure. For instance, the CPAD of the sub-trie shown in Fig. 4 is 3/4, or 75%, which means that roughly 100–75% = 25% of the storage can be saved by bitmap compression. CPAD is relative to the "complexity" of sub-trie structure. Prefixes are more likely to be overlapped with each other (i.e., CPAD is higher) if the sub-trie has more branches or prefix levels.

2.5. Definition: path skip

Path skip is first introduced in the PATRICIA algorithm [1], referred to as *Path Compression*. As is shown in Fig. 5, the left branch of node *A* contains a chain of four internal nodes, each of which has only one genuine child node. Therefore, we can skip these internal nodes during the course of trie-traversing, improving average search performance by reducing the number of nodes that need to be stored or inspected. (We will see in the later section that specific data structure should be introduced to employ path compression.)

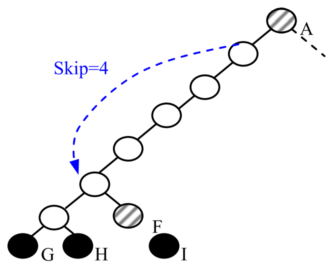


Fig. 5. Path skip is a term associated with the path compression technique. This figure is a part of the trie shown in Fig. 2.

3. Features of real-world route prefix distributions

3.1. Characteristics of real-world IPv4 route prefix distribution

Since IPv6 is still in its initiation period, relatively few IPv6 prefix databases can be utilized for study [11]. What is more, the current distribution of IPv6 route table will be quite different from its future patterns because of the evolving RFC and other regulations. However, it is obvious that the topology of the Internet will not be altered greatly during the migration from IPv4 to IPv6. Therefore, the analysis of IPv4 route prefix distributions is necessary in providing intuitions and clues for designing novel look-up scheme for future IPv6 applications. Although allocated according to different regulations, they still possess some common characteristics. In this section, we first introduce some characteristics of real-world IPv4 route prefix distribution. Then the properties of IPv6 route table will be deduced based on these observations and associated documents.

We have analyzed a large amount of real-world route tables collected from famous route service projects [12,15], and acquired several useful characteristics of the prefix distributions. Due to the space limitation, we pick only two typical ones from these route tables to illustrate our observations, as is described in Table 1.

3.1.1. Distribution on prefix levels

Fig. 6 depicts the prefix distribution on different levels. We can see that the number of prefixes decreases logarithmically with the growth of prefix level (note the logarithmic scale on the *y* axis). Almost all of the prefixes have only five to six levels and the majority (e.g., over 90%) of them are in Level 0 (i.e., they are prefix leaves). This indicates that in the real networks, the subnet hierarchies of the Internet do not have too many levels. Note that prefix level reflects the extent of prefix overlapping. Therefore, the fewer the prefix levels, the fewer overlaps among the prefixes, and therefore the higher bitmap compression ratio is expected to be. Since the network infrastructure/topology of the

Table 1
Two real-world route tables

Name of database	Date	Number of prefixes	Number of next hop
Mae-West [12]	2001–03	33,960	45
RRC06 [15]	2003–11	131,372	35

The Mae-West table from the IPMA project was a very famous table with middle-size; The RRC06 tables are fairly large BGP tables in the real world.

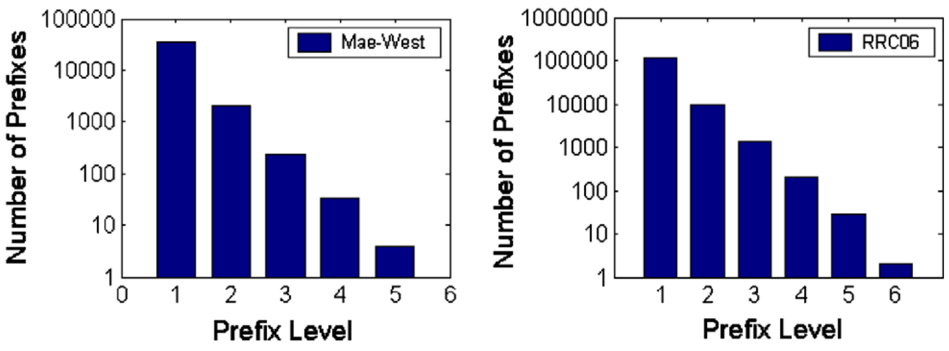


Fig. 6. Prefix distributions on prefix level.

Internet should not be largely altered during the migration from IPv4 to IPv6, this observation is also expected to be useful for IPv6.

3.1.2. CPA Density distributions for different types of sub-tries

Fig. 7 depicts the CPAD distribution for sub-tries with different root and height in 2 route tables. CPAD indicates the potential of bitmap compression and lower CPAD can achieve higher compression ratio. It can be seen that the majority of the sub-tries are fairly compressible. For example, the CPAD of 95% sub-tries are less than 0.3 in the Mae-West table, and 0.4 in the RRC06 table. Moreover, CPAD tends to drop down with the increase of the height and root depth of the sub-trie. However, CPAD also varies considerably among sub-tries even with the same parameters if we have noticed the discrepancy between the "Max" and "Min" bars. Some sub-tries still have relatively high CPAD. This indicates that adopting uniform parameters such as stride for sub-tries might not get good enough performances in terms of memory saving.

3.1.3. Path skip distribution on different depths of the prefix trie

The distributions in Fig. 8 show that path skips of the large route table tend to be shorter than small one. This is easy to understand since more genuine nodes result in less single-child internal nodes that can be path compressed. For example, path skips on Depth 24, which has the largest number of genuine nodes, are distinctively shorter than those on other depths. This observation shows that path compression may not be quite effective for route tables with high genuine node/prefix leaves density (e.g., in the case of the RRC06 table). However, this technique may be useful for future IPv6 applications because low genuine node and prefix leaf density are expected in the case of IPv6, as will be discussed in the next sub-section.

3.2. IPv6 prefix characteristics

In this subsection, we will introduce an initial IPv6 route table and estimate the future IPv6 prefix distribution based on a thorough study of the associated RFCs [16–19] and RIPE documents [11]. Some useful implications for developing IPv6 route lookup algorithms will also be discussed.

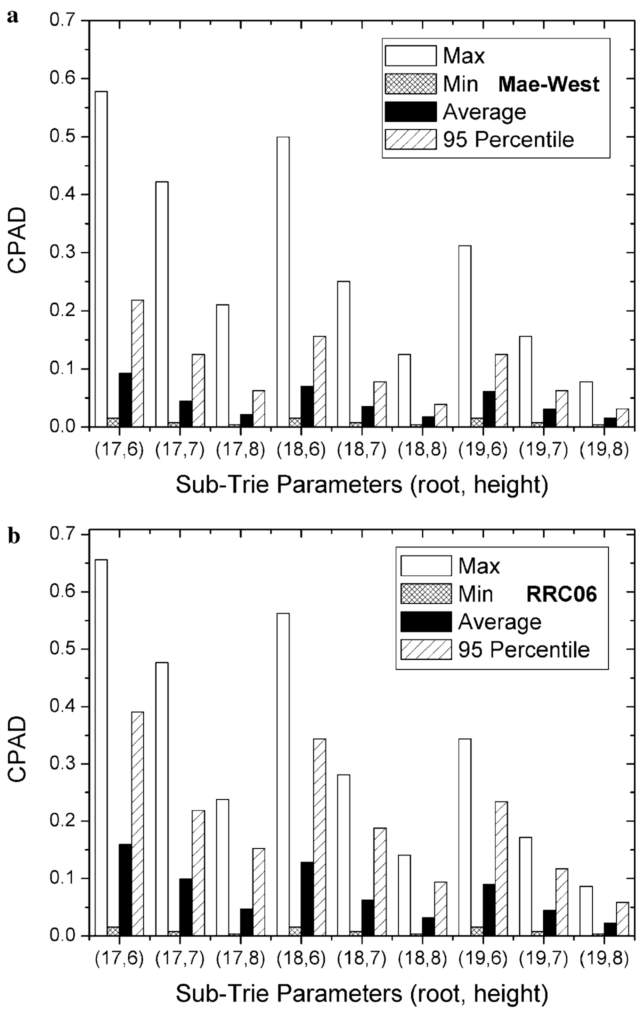


Fig. 7. CPA Density distributions. This figure lists the CPAD distributions for nine kinds of sub-tries along the x axis, rooting from depth 17 to 19 and with a height of 6 to 8, respectively. These sub-tries represent the part of the prefix trie with highest prefix density. (Here we only take sub-tries containing genuine nodes into account.) (a) CPA Density distribution of the Mae-West table. (b) CPA Density distribution of the RRC06 table.

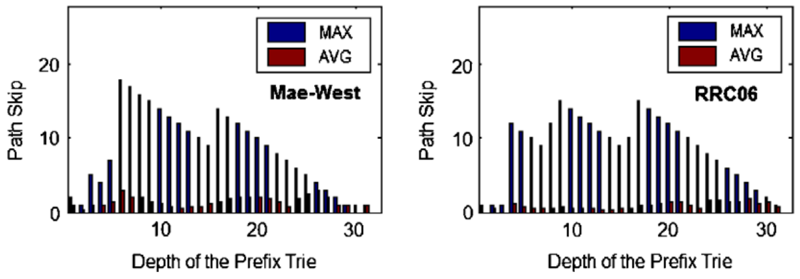


Fig. 8. Path skip distributions on different depths of the trie.

3.2.1. Current (initial) IPv6 prefix distribution

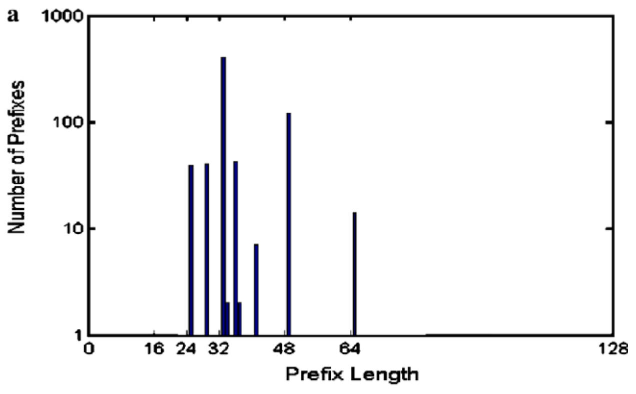
Fig. 9(a) depicts the IPv6 prefix distribution on prefix length of a real-world IPv6 global route table (Route-View IPv6 route table, Data: 2004-10-3, Size: 680 Prefixes[2]. [20]). We can see that the majority are '/32' prefixes, which is referred to as the "initial IPv6 allocation blocks" [11]. As mentioned in [11], this kind of IPv6 address blocks are allocated to the *Local Internet Registries* (LIRs) who: (1) "plan to provide IPv6 connectivity to organizations to which it will assign '/48's by advertising that connectivity through its single aggregated address allocation"; (2) "have a plan for making at least 200 '/48' assignments to other organizations within two years". (Note: the last information is essential for the IPv6 prefix generation, which will be introduced shortly.) Some even shorter prefixes (length from 16-bit to 31-bit) were assigned to high level subscribers according to RFC 2374 [16] before it was replaced by RFC 3587 [19]. In RFC 2374 and RFC 2928 [17], IPv6 address blocks were organized in a complex aggregatable hierarchy which includes the TLA (Top Level Aggregation) '/16' blocks, sub-TLA blocks, NLA (Next Level Aggregation) '/48' blocks, SLA (Site Level Aggregation) '/64' blocks, and the Interface Level address ('/128').

Fig. 9(b) depicts the IPv6 prefix distribution on prefix level, where is very similar to the cases of IPv4. And we can see that there are only four prefix levels (i.e., subnet levels).

3.2.2. Future IPv6 prefix distribution estimation and the IPv6 prefix generator

RFC 3587 (up-to-dated) replaces RFC 2374 and simplifies the aggregatable IPv6 address hierarchy. Now there are only three levels of prefixes: the Global routing prefix (4–48th bits. Note that the 1–3th bits of IPv6 unicast address should be '001'), the subnet ID (49–64th bits), and the interface ID (65–128th bits).

According to RFC 3177 (IAB/IESG recommendation on IPv6 address allocation to sites) [18] and RIPE 267 [11], the IPv6 address blocks should be allocated to subscribes following these rules: (1) '/48' in the general case, except for very large subscribes, which could receive a '/47' or multiple '/48s'; (2) '/64' when it is known that



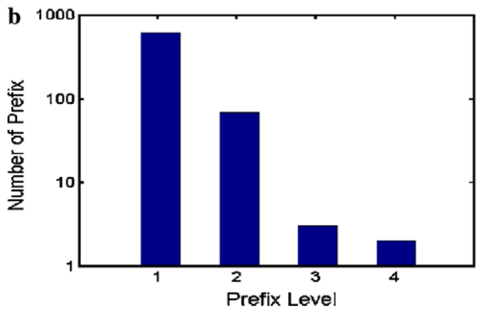


Fig. 9. Real-world (initial) IPv6 prefix distribution on prefix lengths. Note the logarithmic scale on the *y* axis. (a) On Length (b) On Level.

one and only one subnet is needed by design; (3) '/128' when it is ABSOLUTELY known that one and only one device is connecting.

From the related recommendation of RFCs and RIPE documents introduced above, we come to some useful conclusions as follows:

i. It is obvious but important that there is no prefix with length between 64 bit and 128 bit (excluding 64 bit and 128 bit).
ii. The majority of the prefixes should be the '/48s', and '/64s' the secondary majority. Other prefixes would be distinctly fewer than the '/48s' and '/64s'.
iii. Specifically, the number of '/128s' should be tiny, which would be similar with the ratio of the '/32s' in the case of IPv4. Both conclusion No. i and No. iii present a very useful information, that if we do not take account of the minority '/128s', the longest prefix is actually 64 bits.

[2] There are totally 6700 prefixes in the original database, however, only 680 of them are unique (since the database may contain identical route prefix announced by different source routers).

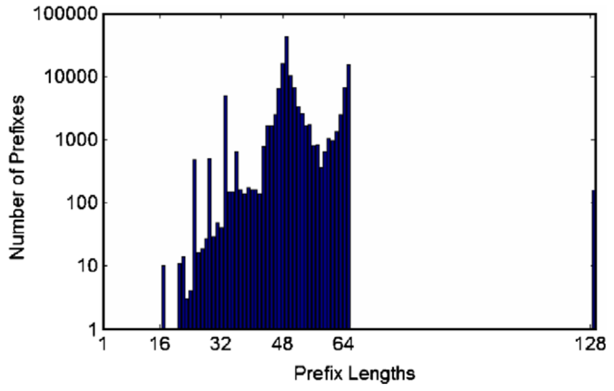


Fig. 10. Artificially generated IPv6 prefixes distributions. According to the above 5 conclusions, we artificially generate 15–30 prefixes from each prefix in the real-world Route View IPv6 route table. There are totally 129483 prefixes in this table, and they are assigned randomly with the Next Hop IP address from 110 distinct IPv6 addresses.

iv. On one hand, the address space of IPv6 increases exponentially from that of IPv4, and the average/overall prefix lengths of IPv6 are distinctively longer than that of IPv4 (also refer to Conclusion ii); on the other hand, the better aggregatability of IPv6 effectively controls the increase of the prefix number, which may even smaller than that of current IPv4. Both these two tend to make the prefix density of IPv6 much smaller than that of IPv4. Hence the IPv6 prefix trie should be more compressible (for both bitmap compression and path compression), according to the analysis in Section 3.

v. Future (or the near future) IPv6 address blocks will be allocated to common subscribers from the current initial LIRs blocks. This is important for IPv6 prefix generating according to current initial allocation prefix.

Based on these conclusions, we develop an IPv6 generator[3] generating subscriber level IPv6 prefixes for IPv6 route lookup algorithm benchmarking. The general function of the generator is described as follows:

Using a real-world “Initially Allocated” IPv6 route table as seed file, the generator analyzes certain characteristics of the table such as prefix number and locations. Based on the analysis, it randomly produces a specific number (assigned by the user) of prefixes from the imported seed prefixes following a given prefix length. Then it will validate the generated prefixes and randomly assigns Next hop IP addresses to the prefixes. This procedure repeats until the pre-defined metrics are met.

Fig. 10 depicts the prefix distribution of a generated IPv6 route table example, which is also used in the performance evaluation of the proposed lookup scheme.

[3] The IPv6 prefix generator and corresponding generated route database are available at: http://zheng_kai.home4u.china.com/V6Gen.htm.

4. A scalable IPv6 lookup scheme via dynamic variable-stride bitmap compression and path compression

Conventional Bitmap Compression algorithms typically employ *fixed-stride tries*. For example, the sub-trie hierarchy of Lulea algorithm is set as three levels and a possible split might use (16, 8, 8) as their sub-trie heights. In the case of Eatherton’s algorithm [7], the stride of each level is fixed to 4. However, fixed stride may suffer from performance degradation in both route table lookup and storage because of the uneven distribution of prefix. As can be seen in our previous analysis, the size of CPA changes a lot even on the same level of trie with the same height. Since variable-sized nodes may lead to memory fragmentation, the storage efficiency is highly depended on the effectiveness of memory allocator [13]. Moreover, a fixed-stride searching scheme can hardly be integrated with the path compression skill, which we found pretty efficient in IPv6.

On the other hand, bitmap compression makes each component of a sub-trie structure correlated with each other. In some cases, updating one prefix may lead to the alternation of the whole sub-trie structure. Moreover, some schemes employ large stride for level one sub-tire in order to reduce the search latency. For example, in Lulea algorithm, the height of the first level sub-trie is 16, which contains $2^{16} = 65536$ edge nodes. Updating such a sub-trie would be very costly.

The motivation of our scheme is to give careful consideration to the discrepancies existed among different part of the prefix trie and to take full advantage of wide word memory architecture. As is shown in Fig. 11, prefix trie is partitioned into sub-tries with variable-heights and packed in fixed-sized wide words. Path compression information among sub-tires is also included when it is necessary. Our heuristic trie-construction algorithm wastes the memory as low as possible and gets rid of the dependence on the effectiveness of memory allocator.

The detailed implementation of wide embedded memory and its applications have been fully discussed in many people’s works [4,14,21]. In our scheme, one wide word may contain one big sub-trie when its CPAD is low enough, or several small sub-tires. Therefore, in addition to the saved memory space, the number of bits to be looked up at a time is increased and not limited by the fixed stride. Moreover, several (successive) sub-tries can be retrieved using a single memory access, where the memory reference times needed for some prefix lookups fall. The combination of path compression further reduces the average search steps. On the other hand, the dependence of the information only exists in single word, which makes the complexity of update operation also greatly decreased.

4.1. Data structure and trie construction of DVSBC-PC

The data structure of each wide word, called as *Word Frame*, is depicted in Fig. 12. Although in our description,

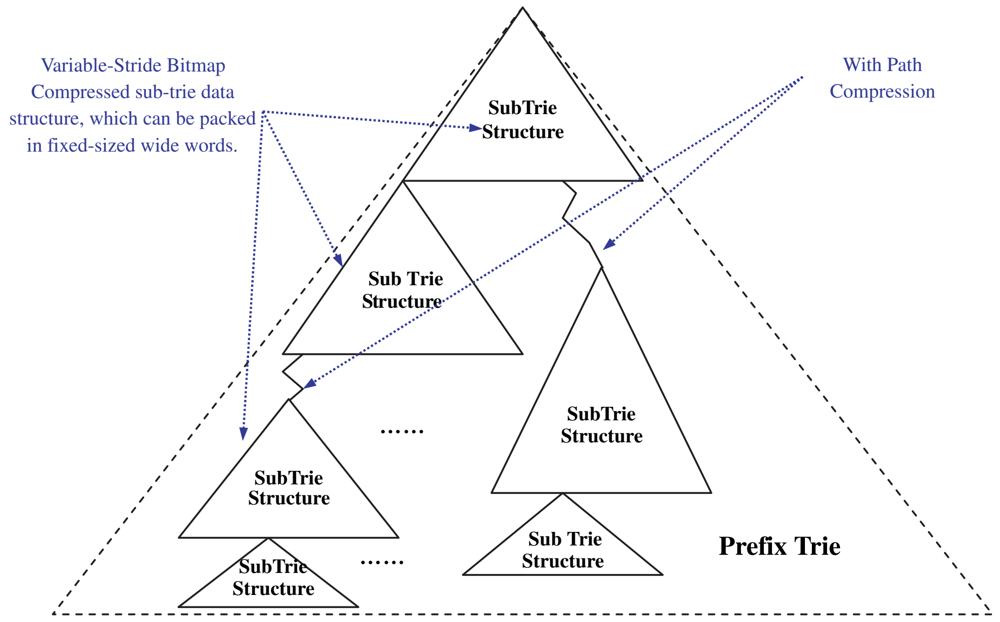


Fig. 11. A demonstration of the idea of DVSBC-PC.

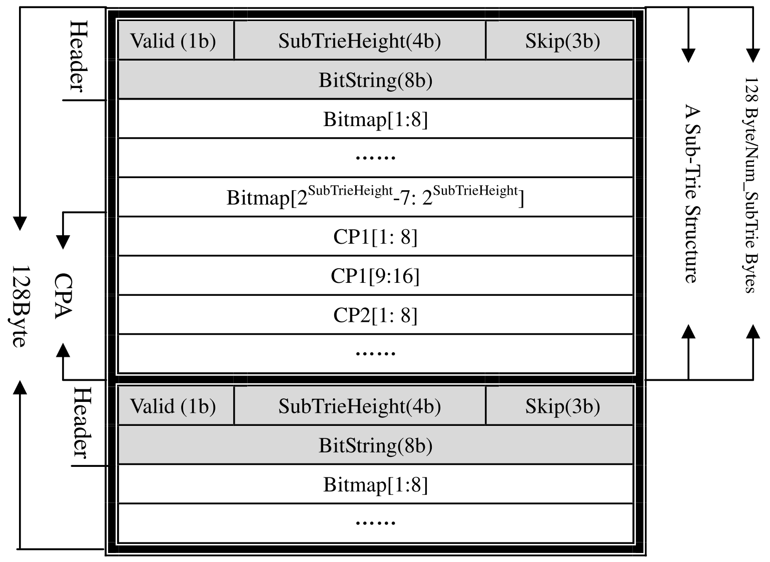


Fig. 12. The data structure of a *Word Frame*.

we use 128-byte as an example, the actual word size (denoted as *WORD_SIZE*) can change with the system implementation. Generally speaking, longer word may produce relatively better search performance because of the decreased average lookup steps. However, blinding enlarging word size also adds much more difficulties in terms of system design, which may not be compensated by the gain in reduced searching time.

Each word frame contains one or more equal-sized sub-tries, which is composed of header information and payload. Therefore, in this example of 128-byte wide word, the memory space used to store sub-trie structure is limited as 128, 64, 32 or 16 bytes, in order to simplify the memory management and leave enough flexibility in the case of update operations. As is shown in Fig. 2, the meaning of each field in Word Frame is as follows:

- **Valid [1 bit]**: Indicating whether this word frame contains valid information.
- **SubTrieHeight [4 bits]**: The height of this sub-trie. If this field is set as k, the length of *Bitmap* will be 2^k bits.
- **Skip[3 bits]** and *BitString [8 bits]*: Information necessary for Path Compression, as has been explained in Section 2.
- **Bitmap[$2^{\text{SubTrieHeight}}$bits]** and **CPA[variable size]**: Information necessary for bitmap compression, as has been explained in Section 2. Each CP within the CPA (i.e., Compressed Pointer Array) is 16 bits, representing

either a Next IP index or a pointer to another sub-trie structure (indicated by the first bit). The size of CPA equals to the total number of “1” in the **Bitmap**. If **Skip** is large than zero, a failure CP is added to the tail of CPA, which is used to return the final lookup result when the search key mismatches with the Bit-String. On the other hand, for a given word length, the height of sub-trie is also confined into certain ranges (denoted as [*MIN*, *MAX*]), according to the actual number of CPA that can be packed into one word. For this example where 128Byte word frames are adopted, the values of *MIN* and *MAX* are 2 and 9[4], respectively.

The algorithm of constructing such a prefix trie is listed in Fig. 13. The outer **while** loop traverses the binary-trie in post order and picks out the next available node *p* for each iteration. If this node has not been visited, the Skip(*p*) operation will perform Path Compression and return the last node (denoted as *next_node* in Fig. 13) in the compressed path. The inner **for** loop tries to find the largest sub-trie that can fit into a wide word. In another word, when the first height *i* is encountered such that the size of the partitioned sub-trie is less than *WORD_SIZE*, this sub-trie is carved out, a free word is allocated, the corresponding CP in the parent node is updated and all the nodes in this sub-trie is labeled as “visited”, except those edge nodes which will be the root nodes of the next level sub-tries.

Note that in some cases, the largest sub-trie may be smaller than 64-byte or even less. The procedure MemAllocate(*subtrie_size*) maintains a list of all the available wide words, including those storing 64-, 32-, or 16-byte sub-trie substructures but still have unallocated spaces. For example, if the largest sub-trie is smaller than 64-byte but bigger than 32-byte, a wide word containing unallocated 64-byte space will be allocated.

4.2. Search algorithm

The search algorithm is listed in Fig. 14. The lookup process is actually to traverse the sub-trie hierarchy. In the initial step, we have the search key (the Destination IP address) and a Compressed Pointer to the root sub-trie structure (the root node of which is the root node of the whole prefix trie). Then, we use the key and the pointer to find the successive word frames and get the corresponding pointers, iteratively, until we finally come to NIP Array containing the corresponding next hop IP address.

[4] Note that 2 bytes are required to represent the Trie Header and 128 minus 2 = 126 bytes = 1008 bits are left for Bitmap and CPA. So the subtrie height is at most $\lfloor \log_2(1008-1) \rfloor = 9$ (i.e. when only one sub-trie structure is stored in the word frame and the size of the CPA is as small as 1). On the other hand, it is straightforward that a sub-trie structure whose SubTrieHeght = 2 can always smaller than 16 bytes.

```
Trie-Construction( )
{
  while (there is a next node in post order traverse)
    p = next node in post order traverse ;
    if (!Visited(p)) then
      next_node = Skip(p);
      for (i = MAX ; i >= MIN ; i--)
        bitmap_size = 2^i/8;
        cpa_size = CPASize(next_node, i);
        subtrie_size = 2 + bitmap_size + cpa_size;
        if (subtrie_size =< WORD_SIZE) then
          pointer = MemAllocate(subtrie_size);
          UpdateCPA(p->parents, pointer);
          VisitTrie(next_node, i-1);
          break;
        endif
      endfor
    endif
  endwhile
}
```

Fig. 13. Algorithm for trie construction.

```
Trie-Search(DIP, p, current_level)
{
  if (p->FirstBit ==1) then
    return p->NIP;
  else
    start = current_level ;
    end = current_level + p->Skip;
    if (p->Skip==0 || Match(p->BitString, DIP[start, end])) then
      start = end + 1;
      end = start + p->SubTrieHeight;
      offset = DecodeBitmap(p->Bitmap, DIP[start, end]);
      return Trie-Search(DIP, p->CPA[offset+1], end);
    else
      return p->FailureCP->NIP;
    endif
  endif
}
DecodeBitmap(bitmap, bit-string)
{
  value = the integer represented by bit-string;
  return the number of "1" in the first value bits of bitmap;
}
```

Fig. 14. Algorithm for trie search.

4.3. A simple example of the DVSBC-PC algorithm

Fig. 15(a) shows the sub-trie hierarchy (which is formed according to the *DVSBC-PC* algorithm) of the prefix trie introduced in Fig. 2. We see that the prefix trie is partitioned into 3 sub-tries. Since path compression is employed, some internal nodes with only one genuine child are omitted (e.g. the three nodes in dashed, in Fig. 15(a)). We find that though the heights of the three sub-tries are 2, 2, and 3, respectively, the CPA size of the three sub-tries are all 4 (pointers), so they can all be stored in a word frame. The corresponding data structure is depicted in Fig. 15(b). Note: (1) in this example, we assume that the length of the IP address is 7 bits and *WORD_SIZE* is 8 bytes. Therefore, the size of sub-trie structure is also constrained as 8 bytes; (2) the example is just for demonstration and in this case very little compression ratio is achieved.

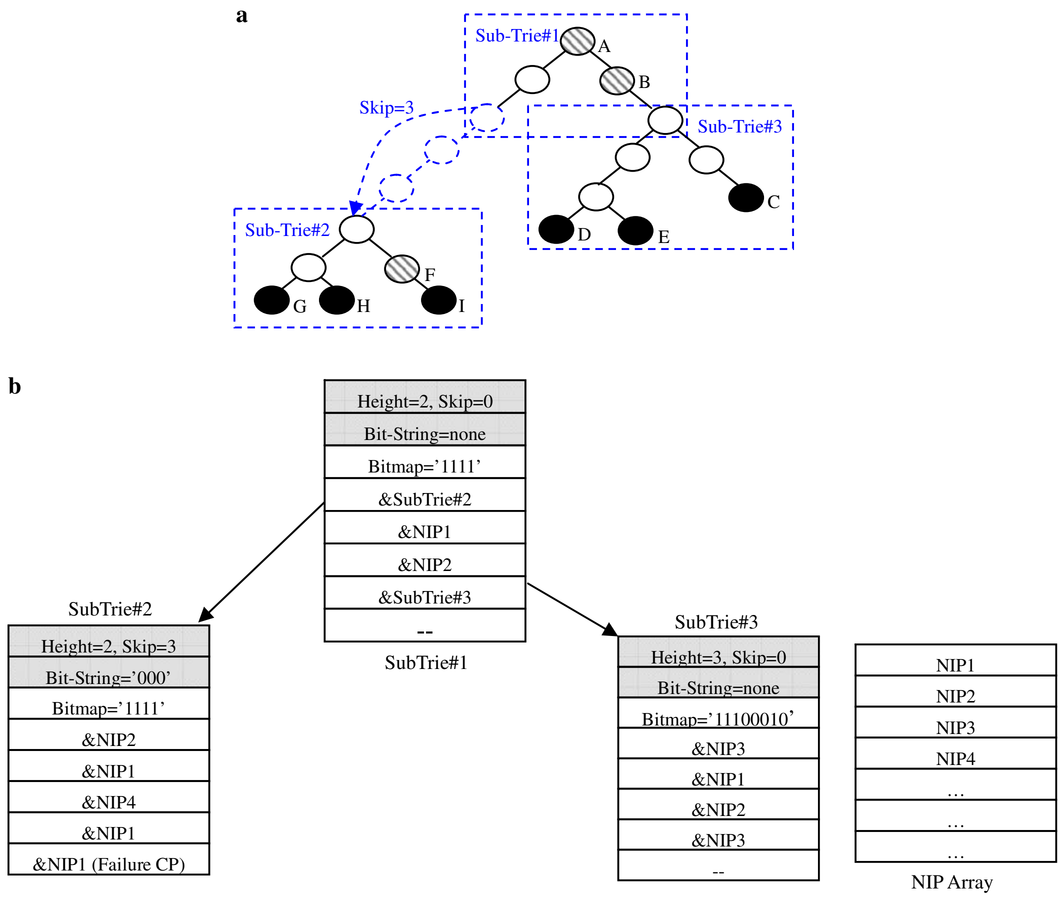


Fig. 15. An example for DVSBC-PC trie constructing. (a) An example for demonstrating the DVSBC-PC algorithm. The prefix trie (introduced by Fig. 2) is partitioned into 3 sub-tries. Note that since path compression is employed, some internal nodes are omitted. (b) The corresponding data structures of the example in sub-figure (a). The memory organization includes 3 sub-trie word frames and the NIP Array. Note that the "skip" value of sub-trie#2 is 3 (>0), so the corresponding word frame contains a failure CP.

Suppose that the key, i.e., the IP address to lookup, is "0000010". The traverse begins with the root sub-trie, i.e., sub-trie#1. We find the corresponding $Height = 2$ and $Skip = 0$, indicating the lookup stride is 2 and no path compression is employed in this step. So we just use the first 2 bits of the key "00" to decode the bitmap-compressed structure, and then we get the offset "0", which implies that we should follow the "$0 + 1 =$ 1st" CP in the CPA, i.e., a pointer points to sub-trie#2.

In the next step, we come to sub-trie#2, and the corresponding $Height = 2$ and $Skip = 3$, which indicates the lookup stride is 2 and 3 nodes are path compressed. So we first use the next 3 bits (i.e., the 3rd to 5th bits) of the IP address, "000", to compare with the Bitstring segment, and result is a match. Therefore, we further use the next 2 bits (i.e., the 6^{th} to 7^{th} bits), "10", to decode the bitmap-compressed structure: Since the string "10" represent integer "2", and there are two '1's in the first 2 bits of Bitmap, so we get the offset "2", which implies that we should follow the "$2 + 1 =$ 3rd" CP in the CPA. Finally, the traverse terminates when we come to the NIP Array and the corresponding lookup result is NIP4.

4.4. The CAM skipping scheme – adopting CAM to gear up the lookup performance

A dedicated CAM mechanism based on the following two heuristics can be introduced to further improve the lookup performance.

Let $k_{\min}$ be the length of the shortest prefixes. According to the analysis in section 3 the numbers of genuine nodes on depth $k_{\min}$ or slightly larger than $k_{\min}$ are very tiny. And note that the value of $k_{\min}$ is about 16 in the case of IPv6. If we introduce a k'-bit-wide CAM to perform the first level lookup, where $k' \geqslant k_{\min}$, the number of memory accesses for each lookup can be greatly reduced, since the stride of first level lookup is then increased to k'.
The number of 128 bit prefixes (or 32 bit prefixes in the case of IPv4) is small; again if we introduce a 128 bit-(or 32 bit-, in the case of IPv4) Binary CAM (BCAM) for these prefixes alone, the worst-case performance can also be improved.

The CAM mechanism employed for the first level lookup is as depicted in Fig. 16, which includes three partitions:

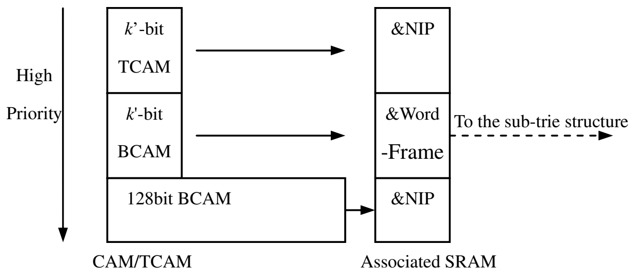


Fig. 16. The CAM organisation. Note that if $k' = k_{\min}$, the k'-bit TCAM can be omitted.

- A k'-bit wide TCAM (i.e., Ternary CAM) is used to store the prefixes no longer than k'-bit, which is similar to the conventional TCAM-based lookup scheme. And the pointers to the Word Frame containing the corresponding Next-hop IP addresses are stored in the associated SRAM.
- A k'-bit[5] wide BCAM is used for the prefixes between k'-bit and 64-bit. Each genuine node on depth k' takes a BCAM entry, and the associated SRAM stores the pointer to the corresponding sub-trie structure rooting at this genuine node.
- A 128-bit wide BCAM is used to store the 128-bit prefixes alone.

Now, given a key to search, it will be first sent to the mixed-CAM mechanism for a match. Then the matching result with the highest priority (as depicted in Fig. 16) is returned. The result includes the pointer to a word frame containing either the associated next-hop IP address (for prefixes less than k' bits or exactly 128 bits) or the sub-trie structure whose root node exactly matches the first k'th bits of the key. Then the following process is similar with the original scheme.

5. Performance evaluation

To evaluate the performance of our scheme, we construct the forwarding tables for a large IPv4 real-world table (RRC06) and one synthesized IPv6 table introduced in Section 3 using *DVSBC-PC* and *DVSBC-PC-CAM*. In the latter scheme, we set $k' = 12$ for IPv4 and $k' = 30$ for IPv6. Since the evaluation results for the middle-sized IPMA are similar with those of RRC06, we only show the results for RRC06 due to space limitation.

For the purpose of comparison, the famous multi-bit trie algorithm, *DIR-21-3-8* [5], the *Tree Bitmap* Compression schemes [7], and conventional TCAM-based lookup engines are also investigated. Since *Lulea* algorithm [6] cannot be used for large route tables due to the limited pointer size in the original configurations, we do not take it into consideration in the following descriptions.

5.1. Memory requirements

The comparison of memory consumption for these different lookup schemes is shown in Table 2. The forwarding table constructed by *DVSBC-PC* is the smallest one, and can be easily extended to support large route tables or long IP address. This demonstrates that our algorithm makes more use of the characteristics of route table and achieves a higher compression ratio than the other schemes. Compared with conventional TCAM lookup engines, *DVSBC-PC-CAM* consumes very tiny TCAM that can be easily embedded into chips-like network processor. Although the overall memory requirement is slightly higher than *DVSBV-PC* for some route tables, the search time and update complexity are greatly reduced as explained in later subsections.

5.2. Lookup performance

Table 3 shows the number of memory accesses for different lookup schemes. Although the IPv6 prefixes are generally much longer than IPv4, migration from IPv4 to IPv6 only causes less than 2 additional memory accesses on the average in the two proposed schemes. This indicates that both path compression and variable-stride bitmap-compression perform much better in IPv6 than IPv4, since the prefix density of the former is much less than the latter. On the other hand, the searching steps of *DVSBC-PC-CAM* shrink to less than 2/3 of that of *DVSBC-PC* for both the average and worst cases, demonstrating the effectiveness of skipping technique when CAM is introduced.

Suppose that we adopt the state-of-the-art 5ns SRAM. Then a single *DVSBC-PC-CAM* search engine achieves 1s/(9*5)ns $\approx$ 22MSPS (Million Searches Per Second) for IPv6, or 1s/(5*5)ns = 40MSPS for IPv4 even in the worst case, which can support 10 Gbps wire-speed packet forwarding for back-to-back smallest packets[6]. Although the performance of *DVSBC-PC* scheme is relatively lower, the average searching performance is also sufficient for 10 Gbps links.

Note that although *DIR-21-3-8* outperforms the proposed scheme in terms of searching steps, the huge memory consumption makes it only suitable for long latency SDRAM and can hardly be scaled to support IPv6. The conventional TCAM also achieves higher search throughput at the expense of higher cost and power consumption.

[5] The selection of value k' is a tradeoff between the lookup performance and memory (CAM) consumption. The larger the value of k' is, the more the number of memory accesses can be reduced, however the more the consumption of CAM is required.

[6] The minimum-sized packet is 60 bytes (40 bytes IPv6 header plus 20 bytes TCP header) for IPv6 and 40 bytes for IPv4. Hence the maximum packet rate should be 10 Gbit/(60*8 bit) $\approx$ 20.8Mpps (Packets Per Second) for IPv6 and 10 Gbit/(40*8 bit) $\approx$ 31.3Mpps for IPv4.

Table 2
Memory requirements for different lookup schemes

Schemes	IPv4		IPv6	
	TCAM	RAM	TCAM	RAM
DVSBC-PC	–	334 KB	–	447 KB
DIR-21-3-8	–	>9 MB	–	N/A
Tree Bitmap	–	808 KB	–	630 KB
DVSBC-PC-CAM[a]	0.9 KB	338 KB	2.1 KB	439 KB
Conventional TCAM	511 KB	511 KB	2 MB	2 MB

N/A, not available.
[a] Assume that 1K TCAM = 2K BCAM.

Table 3
Lookup performance comparisons

Schemes	IPv4		IPv6	
	Worst	Avg.	Worst	Avg.
DVSBC-PC	8	7.17	14	8.3
DIR-21-3-8	3	1.64	N/A	N/A
Tree Bitmap	11 (8 + 3)	8.58	35 (32 + 3)	14.9
DVSBC-C-CAM	5	4.33	9	5.1
Conventional TCAM	1	1	2	2

N/A, not available.

5.3. Updating performance

For the DVSBC-PC scheme, since it is developed based on the trie, prefix insertion or deletion operation is similar with those of the binary trie algorithm which can be easily implemented in a incremental way. The difference is that compressed sub-trie structures, instead of trie node structures, are stored. And inserting/deleting a prefix may also lead to an increase/decrease of the memory requirement of the corresponding compressed sub-trie structure. Sometimes the previously allocated memory space may be no longer enough to accommodate the modified sub-trie structure, or sometimes the previously allocated memory space may be "too large" for the modified sub-trie structure which results in waste of memory. In these cases, additional memory accesses are needed to adjust the memory organization:

When a prefix insertion/modification leads to overflow, i.e., the previously allocated memory space is no longer enough to accommodate the modified sub-trie structure, then:

i. If the allocated size is no more than 128 bytes (i.e., 64, 32, or 16 bytes), a new memory space (in another word frame) is allocated to the sub-trie structure, the size of which is doubled. In this case two memory accesses are needed additionally, one to write the new word frame and the other to modify the corresponding pointer in its parent sub-trie.

ii. If the allocated size is already 128 bytes (the largest size for a sub-trie structure), the original sub-trie would be splitted into three sub-tries, as shown in Fig. 17. In this case, six memory accesses are needed additionally, three for new word frame write-in, respectively, and the other three for the corresponding pointers modifications.

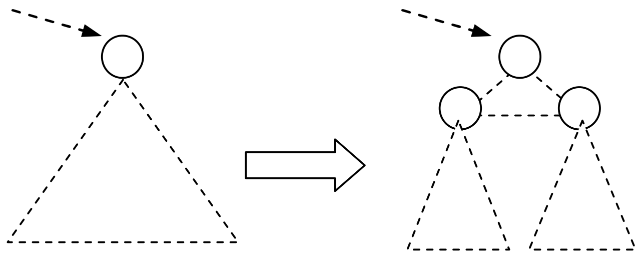


Fig. 17. Sub-trie splitting when overflow occurs.

When the prefix deletion/modification results in the actual size of the sub-trie structure is smaller than half of the allocated memory space (which is larger than 16 bytes), a new and half-sized memory space would be allocated to the sub-trie structure. In this case, two additional memory accesses are needed, one to move the sub-trie structure and the other to modify the corresponding pointer in its parent sub-trie.

For the DVSBC-PC-CAM scheme, when updating the 128-bit prefixes or the prefixes shorter than k' bits, it is actually the same with conventional CAM-based schemes, so incremental update can be easily achieved. When updating the prefixes between k' bits and 64 bits, it is similar with the DVSBC-PC scheme, except that in some cases one additional BCAM access may be needed.

In summary, the update process of both schemes can be performed in an incremental way. No any counter updating or mass structure rebuilding is needed at all.

5.4. Scalability and flexibility

The main intention of this paper is not to present a prototype of an IPv6 lookup scheme implementation with specific design parameters, but to utilize the proposed word frame and dynamic variable-stride lookup scheme to achieve both storage and lookup efficiency for IPv6 address lookup. Though we have defined a 128-byte word frame in a specific way, the lookup scheme is actually flexible.

- In the proposed *Word Frame* structure, we use 16-bit-wide pointers, the first 13 bits of which is used for word frame addressing. So the prototype supports up to $2^{13} = 8192$ word frames, i.e., about 1 Mbytes memories. Note that only less than 500 Kbytes are needed for a route table with more than 130K prefixes. For extremely large route tables, if exists in the future, we may enlarge the pointer size accordingly. For instance we can raise the pointer size to 18 bits, and then support up to 4 Mbytes memories. However, this results in about 18/16–100% = 13% memory requirement overhead, which implies that the pointer size should be carefully chosen to tradeoff between scalability and memory requirement.

- The size of *Word Frame* can be adjusted according to the system environment. Note that wider memory word may generally result in relatively better lookup performance since the upper bound of each lookup stride would be larger. However, blindly increasing word wide may also incur waste of memory because more pointer bits may be needed for addressing within each *Word Frame*.
- The proposed lookup scheme still works even though on-chip wide-word technique may not be available. In this case, a data-bus-extended off-chip memory module with multiple SRAM chips can be adopted alternatively.
- According to the analysis in Sections V.B, a single DVSBC-PC-CAM engine provides over 22M IPv6 lookups per second, supporting 10 Gbps line rate even in the worst case. Moreover, actually, scalable throughput can be achieved when pipeline or parallel fashion with multiple memory units is adopted. For instance, if the sub-trie structures of different levels are stored in different physical memory units, respectively, a pipeline skill can be employed and results in speedup of the lookup throughput.

6. Conclusion

The significantly increased address length of IPv6 poses a great challenge on wire-speed packet forwarding for high-end routing devices. In this paper, we propose a novel IPv6 route lookup scheme based on some implications acquired from a thorough study of the real-world route tables and associative documents for IPv6 prefix allocation. Our scheme combines the bitmap compression with path compression to fully exploit the high compressibility of IPv6 route table. Taking the uneven distribution of prefix trie into consideration, variable-stride trie partitioning is employed to further reduce the memory requirement and searching steps. We also develop a data structure called word frame and associated trie construction algorithm. When implemented in the newly emerged technique named wide word, fast incremental update can also be achieved.

The experimental results show that only some hundreds Kbytes memory is needed for a synthesized large IPv6 route table containing 130K prefixes. A single lookup engine with state-of-the-art 5ns SRAM achieves more than 22 million lookups per second even in the worst case, which can support wire-speed packet forwarding for 10 Gbps links.

The ongoing researches of our team will cover: (1) further improving the compression ratio and saving more memory; (2) extending the proposed scheme to support applications that require multi-field searching, such as Packet Classification.

References

[1] D.R. Morrison, PATRICIA – practical algorithm to retrieve information coded in alphanumeric, J. ACM 15 (4) (1968) 514–534.
[2] S. Nilsson, G. Karlsson, IP-address lookup using LC-tries, IEEE J. Select. Areas Commun. 17 (6) (1999).
[3] A. Donnelly, T. Deegan, IP route lookup as string matching, in: Proceedings of IEEE Local Computer Networks, 2000, pp. 589–595.
[4] J. van Luntereb, Searching very large routing tables in wide embedded memory, in: Proceedings of IEEE GLOBECOM, vol. 3, 2001, pp.1615–1619.
[5] P. Gupta, S. Lin, N. McKeown, Routing lookups in hardware at memory access speed, in: Proceedings of IEEE INFOCOM'98, 1998, pp. 1240–1247.
[6] M. Degermark, A. Brodnik, S. Carlsson, S. Pink, Small forwarding tables for fast routing lookups, in: Proceedings of ACM/SIGCOMM'97, 1997, pp. 3–14.
[7] W. Eatherton, Hardware-based internet protocol prefix lookups, Washington University Electrical Engineering Department (MS Thesis), 1999.
[8] D.E. Taylor, J.S. Turner, J.W. Lockwood, T.S. Sproull, D.B. Parlour, Scalable IP lookup for Internet routers, IEEE J. Selected Areas Commun. 21 (4) (2003) 522–534.
[9] CYRESS. http://www.cypress.com/.
[10] IDT. http://www.idt.com/products/.
[11] RIPE 267: APNIC, ARIN, RIPE NCC, IPv6 Address Allocation and Assignment Policy, Document ID: ripe-267, January 2003, available at: http://www.ripe.net/ripe/docs/ipv6policy.html.
[12] Database of the Mae-West router from the IPMA Project (A joint effort of the University of Michigan and Merit Network) http://www.merit.edu/ipma.
[13] S. Sikka, G. Varghese, Memory-efficient state lookups with fast updates, ACM SIGCOMM Comput. Commun. Rev. 30 (4) (2000) 335–347.
[14] N. Tuck, T. Sherwood, B. Calder, G. Varghese, Deterministic memory-efficient string matching algorithms for intrusion detection, in: Proceedings of IEEE INFOCOM'04, vol. 4. Hong Kong, 2004, pp. 2628–2639.
[15] Database of the RRC06 router from the RRCC Project (Routing Registry Consistency Check Project), available at: http://www.ripe.net/rrcc/.
[16] RFC 2374: R. Hinden, M. O'Dell, S. Deering, An IPv6 Aggregatable Global Unicast Address Format, 1998, available at: ftp://ftp.ripe.net/rfc/rfc2374.txt.
[17] RFC 2928: R. Hinden, S. Deering, R. Fink, T. Hain, Initial IPv6 Sub-TLA ID Assignments, 2000, available at: ftp://ftp.ripe.net/rfc/rfc2928.txt.
[18] RFC 3177: IAB, IESG, IAB/IESG Recommendations on IPv6 Address. 2001, available at: ftp://ftp.ripe.net/rfc/rfc3177.txt.
[19] RFC 3587: R. Hinden, S. Deering, E. Nordmark, IPv6 Global Unicast Address Format, 2003, available at: ftp://ftp.ripe.net/rfc/rfc3587.txt.
[20] Route-View v6 database, available at: http://archive.routeviews.org/route-views6/bgpdata/.
[21] T. Sherwood G. Varghese, B. CalderA, Pipelined memory architecture for high throughput network processors, in: Proceedings of the 30th International Symposium on Computer Architecture (ISCA), 2003, pp. 288–299.

Kai Zheng received the Master degree in computer science in 2003 from Tsinghua University, China. Now he is a Ph.D candidate in the same university. His current researches focus on high performance IP route lookup, packet classification and network security related to Intrusion Detections System and other string-matching associated network architectures.

Zhen Liu received the Master degree in computer science in 2004, in Tsinghua University, China. Now she is a Ph.D candidate in the same university. Her current researches focus on the network processor architecture as well as its memory hierarchy design.

Bin Liu (M'03) received the M.S. and Ph.D. degree both in computer science and engineering from North-Western Poly-Technical University, Xi'an, China in 1988 and 1993, respectively.From 1993 to 1995 he was a postdoctoral research fellow in the National Key Lab. of SPC and Switching Technologies, Beijing University of Post and Telecommunications. In 1995 he transferred to Dept. of Computer Science, Tsinghua University as an associated professor, where he mainly focused on multimedia networking including ATM switching technology and Internet infrastructure. He is now the director of the Lab. of Broadband Networking Technology and a full professor of Computer Science and Technology at Tsinghua University, Beijing, since 1999. He has been the PI of many projects related to high-speed switches, core routers, network processors as well as multimedia access gateways. His current research areas include high performance switches/routers, network security, network processors and traffic engineering. He holds 11 Chinese patents and has published more then 100 papers in the above fields. He received lots of academic and honorary awards from the home and the abroad. Prof. Liu served as TPC member for INFOCOM 2005-2006, HRSR'05 (also Panel Chair), ICCCN'05. He is now a Guest Editor for the IEEE Journal of Selected Areas- Special Issue on the High Speed Network Security and a member of Communications Security Technical subCommittee (CSTsC), ComSoc.