DEVELOPMENT FIELD

A Multi-level Index Lookup Algorithm and Its Implementation

Yan Xincheng
(Central Academy of ZTE Corporation, Nanjing 210012, China)

Abstract:
Routing lookup is an important technology for IP transmission network and IP-based packet switching network. Although the multi-branch trie tree lookup-algorithm is a fast and efficient routing lookup algorithm, it consumes large memory. The multi-level index lookup-algorithm, as an improved multi-branch trie tree lookup algorithm, is proposed. It expands the original two-level index to multi-level index and introduces a continuous flag bit storage method that is able to extremely reduce memory overhead of routing index table while not obviously affecting the lookup efficiency.

Routing lookup is an important technology for IP transmission network and IP-based packet switching network. The multi-branch trie tree lookup-algorithm[1] is a fast and efficient routing lookup algorithm, which looks up the index table, one after another, of a trie tree structure according to a certain bit segment in an IP address to find the corresponding route. An improved algorithm is proposed that introduces an index compression mechanism and is referred to as "two-level multi-branch compression trie tree algorithm[2]".

1 Best-match Rule of Routing Lookup

There are 4 assumptions followed by their respective mathematical expressions:

(1) Assume that an IP address *a* falls within the index range of route *A*. Then *A* is the route of *a*, i.e., *A* matches *a*. The mathematical expression is:

if $a \,\&\, A_{pfx\_len} = A_{addr}$, then $a \in A$

Here, A_{addr} and A_{pfx_len} are the network prefix and mask of *A*. $a \in A$ means that *a* belongs to *A*, i.e., *A* is the route of *a*. The default route is one of any IP addresses.

(2) Assume the index range of route *A* includes the index range of route *B*. Then *A* is the sub-match route of *B*. The mathematical expression is:

if $A_{pfx\_len} > B_{pfx\_len}$ and $B_{addr} \,\&\, A_{pfx\_len} = A_{addr}$, then $B \subset A$

Here, $B \subset A$ means *B* is a subset of *A*, i.e., *A* includes *B*. Any route is the subset of the default route.

(3) Assume the route *A* has the longest mask among all routes of address *a*, then *A* is the best-match route of *a*. That is, the longest-mask route or the route that performs effective transfer for *a*. When *a* performs route lookup, the result should be *A*. The mathematical expression is:

if $A_{pfx\_len} = \max(X_{pfx\_len}), X \in R_a = \{a \in A\}$, then $a \in = A$

Here, R_a is the set of all routes for a in the routing table. If $a \in A$, there must be $A \in R_a$. $a \in = A$ then indicates that *A* is the best route for *a*.

(4) Assume the route *A* has the longest mask among all sub-match routes of route *B*. Then *A* is the longest-mask sub-match route of *B*. The mathematical expression is:

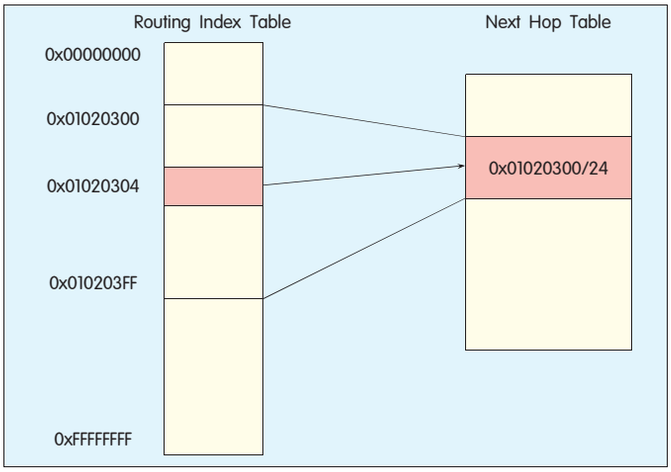
if $A_{pfx\_len} = \max(X_{pfx\_len}), X \in \{B \subset X\}$, then $B \subset = A$

Here, $B \subset = A$ indicates that *A* is the longest-mask sub-match route of *B*.

The best-match rule of routing lookup is referred to as the longest-mask match rule. To search for a route corresponding to an address, if several routes are there to be used, the one with the longest mask is selected.

2 Principle of Segmented Index Table

The segmented index table uses the mapping space of an address to directly locate the routing table and uses index to indicate the prefix address space of a route, thus establishing one-to-many relationship between address and index. If the memory issue is taken out of account, we can set up an index array to indicate the IPv4 all-address space. This index table is then called "All-IP Address Mapping Index Table", as shown in Figure 1. The routing table is divided into two parts. The route index table on the left is responsible for fast routing lookup. The next hop table on the right stores transfer information such as the next hop of a route, which is usually referred to as the routing table. The index table is composed of 4 294 967 296 index entries, indicating IPv4 addresses from 0x00000000 through 0xFFFFFFFF by turn. Every entry of the index table is used to store the pointer of the corresponding next hop table. To use IP address to search for a route, take the IP address as the array subscript in the index table to retrieve the corresponding routing table entry. For example, to find the route for the IP address 1.2.3.4, the 0x01020304th entry is taken out from the index table. This entry corresponds to the route 0x01020300/24 through next hop table. Each index table entry uniquely corresponds to a routing table (in the case there is no corresponding route, the pointer is "Null"), a route however corresponds to several index



▲Figure 1. All-IP address mapping index table.

table entries. In Figure 1, route 0x01020300/24 corresponds to 256 indexes from 0x01020300 through 0x010203FF.

In the all-IP address mapping index table, the number of index entries to which a route corresponds depends on the route's mask length. If the route's mask is *pfx_len*, then the route corresponds to 2^{32-pfx_len} entries of index table.

If an index corresponds to several routes, that is, these routes have overlapping index ranges, then the index will select the route with the longest mask. In Figure 2, the routes 0x01020300/24, 0x01020300/16 and 0x01020304/32 in the routing table are routes of address 0x01020304, and the third route will be selected as it has the longest mask.

3 Two-level Index Table

3.1 Structure of Two-level Index Table

The all-IP address mapping index table undoubtedly has high efficiency in routing lookup (it's accurate "positioning" to be exact, rather than "lookup"). However, the memory consumption is huge. As the index table has 4G entries and each entry stores 4 bytes of next hop table pointer, the whole index table consumes 16 GB of memory, which is beyond feasibility for the time. Virtually the routing table adopts level-by-level indexing, that is, a multi-level index table links to and locates a route. This algorithm is derived from the multi-branch trie tree lookup algorithm proposed by Gupta. Huang and others then introduced index compression mechanism into that algorithm and that index mechanism is usually referred to as "two-level index".

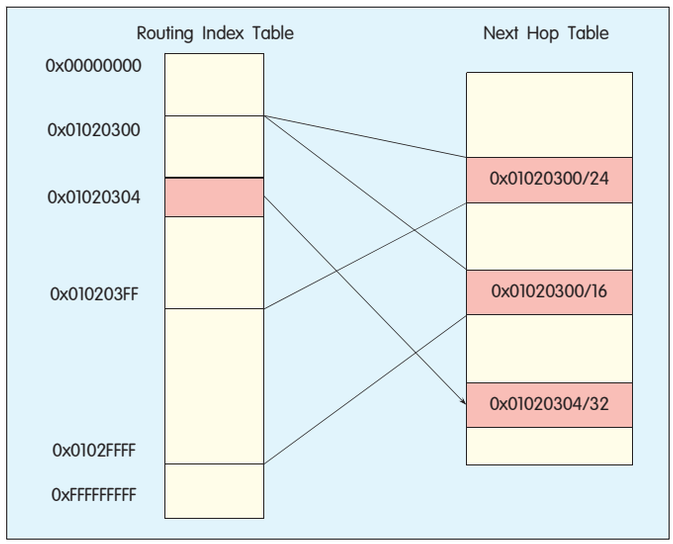
The two-level index table is made up of two levels of index, as shown in Figure 3. Level-one index features invariable step length and its entries comprise of two parts: pointer and flag. The pointer takes the multiplex mode, i.e., the level one index table may either point to the routing table directly or point to the level-two index table, which then points to the routing table. The flag indicates the step length of the level-two index table. If flag is 0, the index table points to the routing table directly. The level-two index employs index compression mechanism and selects proper step length based on route's mask length. (The length of IP address that can be represented by certain level of index table is called the step length of this level of index table). If the step length of certain level of index table is n, then the index table has 2^n index table entries.

For easy description, we use the 24-8 mode to demonstrate this point. That is, the step length of the level-one index table is 24 while the maximum step length of the level-two index table is 8. We'll again try searching for address 0x01020304 as an example. Since the step length of the level-one index table is 24, we take the first 3 bytes 0x010203 of the address as the array subscript of the level-one index table. The corresponding flag of index table is 6, indicating that the pointer entry of level one index table doesn't point to the routing table directly, but to a level-two index table with the step length of 6. Therefore, the owest 1 byte will locate its offset in the level-two index table. Since the step length in question is 6, the offset value is the high 6 bits of the last 1 byte, namely, 0x04>>2, and what stored at this position is the pointer of the corresponding route. As the total step length of two levels of index table is 30, the level-two index points to a route with the maximum mask of 30 in length. That is to say, the total step length of index table must not be shorter than the route's mask. If the mask length of a route is shorter than step length of the level-one index, there is no need to add a level-two index table and the corresponding flag in the level-one index table is 0. That is indicating the index entry points directly to the routing table.

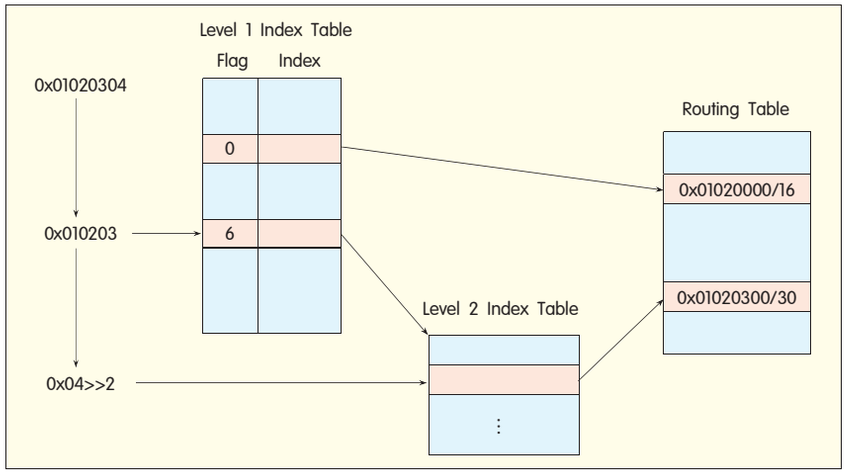
3.2 Memory Overhead of Two-Level Index

We'll take the 20-12 two-level index table as example. If the routing table supports up to 1 K routes, then the whole index table will consume maximal. memory in the case that each route has a corresponding largest two-level index table pointing to it and in that case the index table will need $2^{20}+2^{12}\times 1$ K=5 M index entries. Obviously, this number makes enormous improvement over the 4G entries mentioned before for all-IP address mapping index table. Now we know that level-by-level indexing saves memory space greatly, and this is because:

- the mask length of route is not always 32;



▲Figure 2. Schematic diagram of longest-mask matches.



▲Figure 3. Schematic diagram of routing lookup with two-level index table.

• number of route entries supported by program is limited.

Why the level-by-level index table saves memory space? Because it doesn't have to map all IP address space but just part of it according to the route in the routing table. Thus, the sample space of address is associated to some extent to the sample space of route. Should a two-level index table be used to represent an IPv4 all address space, it's necessary that each level-one index be attached with one largest level-two index table. That totals to $2^{20}+2^{12}\times2^{20}$=4 097 M index table entries. In a word, the level-by-level index table saves memory because of its partial mapping of the IP address.

The level-by-level index table follows suit in using IP address to locate the routing table directly, with different number of attempts though. The all-IP address mapping needs just one attempt of location while the two-level index needs two at most. However, it's still direct location and the lookup efficiency is not obviously degraded.

Nevertheless, the two-level index table sometimes is not adaptable. Such index structure still needs large memory space. To represent an address route with 32-bit mask, memory space of $2^{12}\times4$=16 K bytes should be applied for as the second level of index table, which can result in great memory waste. Besides, as the second level of index adopts compression mechanism, which saves on memory though, it's hard to predict the size of the index table. In addition, it is difficult to fit to the memory management mode of memory pool set technology.

4 Multi-level Index Table

4.1 Structure of Multi-level Index Table

As the multi-layer multi-branch trie tree, multi-level index also searches for routing table by segmented address location. It has made the following improvements to the memory drawback in two-level index.

• Locate routing table by serially connecting multi-level index tables. This algorithm makes use of more specific IP address space and is able to effectively reduce memory consumption of index table, according to analysis covered in the previous section.

• Change the 4-byte pointer of index entry into 2-byte array label.

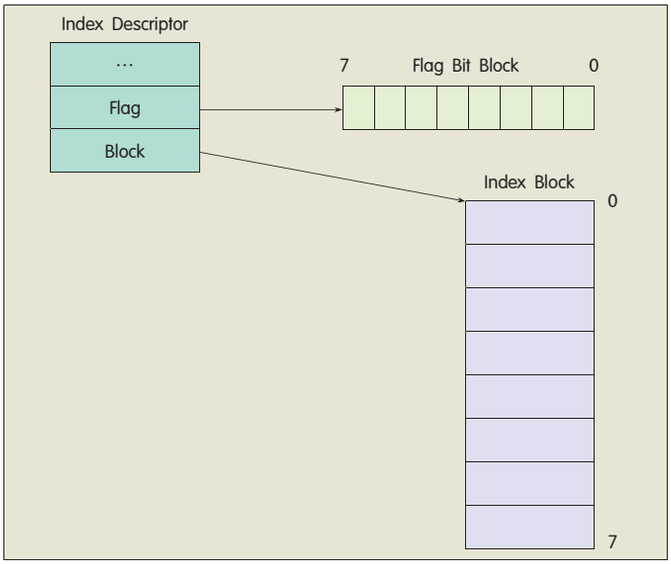
• The flag field in two-level index table is defined to contain 1 byte only, it virtually occupies 4 bytes because of the byte alignment need. The actual memory consumption of flag field may be decreased if the flag bit is stored continuously.

• Give up index compression technique, that is, make index step length invariable. The index compression technique doesn't really decrease memory overhead that much if the index step length is short, it will complicate algorithm instead. Meanwhile, the index blocks with invariable step length can be suitable for any memory management mode.

In a multi-level index table, indices at the same level have the same step length. Indices at different levels may have different step lengths. Total step length of all index tables at different levels for an IPv4 routing table is 32. The entry of a multi-level index table is called "index entry", which is a 16-bit serial number. The serial number can be multiplexed and it means the array subscript of routing table or the array subscript of the index descriptor table of next-level index. Here the routing table is a narrow-sense one that stores information framework array, such as next hop. The array made up of index entries is called "index block" that corresponds to the index table in two-level index algorithm and represents several specific bits in route prefix. What makes it different is that the step length of each level in a multi-level index is fixed (while the number of entries of the second level index in the two-level index algorithm is not fixed). Therefore, each level of index block has fixed size, thus the memory space consumed by each index block is predictable and the code complexity is reduced.

Since the serial number of index entry is multiplexing, one flag bit is needed to distinguish index entry meaning. If this flag exists as a member of the index entry, it will occupy at least one byte (more if considering byte alignment). To save space, we associate each index block to a flag bit block as shown in Figure 4. Each bit of flag bit block logically corresponds to an entry in index block to distinguish content of index entry attribute. If there are 8 entries of index block, the corresponding flag bit block is 1 byte in size, that is, 8 bits.

To strengthen the association between flag bit block and index block, we put their start addresses into the same one framework, which is called "index descriptor". As shown in Figure 4, the two pointers of index descriptor point to the flag bit block and index block respectively. Each bit in the flag bit block corresponds to one entry in the index block. For example, meaning of the fifth index entry in the index block depends on the fifth bit in the flag bit block. With the introduction of the index descriptor, the index block and its corresponding flag bit block can be located—but indirectly with the index descriptor. The flag bit block is an array stored in the form of bytes. If the offset, i.e., array subscript of address in index block, is known, we can divide the subscript by 8 to obtain the subscript of corresponding flag bit block. Then we divide the subscript by 8



▲Figure 4. Correlation between flag bit and index.

and keep the residue to get the corresponding bits.

Each level of index may possibly contain multiple descriptors that are exist in the form of arrays and are called "index descriptor table". Each level of index has a unique index descriptor table that is used to store all descriptors of this level. The index descriptor can be used to locate index block. Therefore, we may record the subscript of descriptor, to which the index block corresponds, in index descriptor table (as shown in Figure 5), when it comes to connect to the next level of index block. The multi-level index algorithm locates a route by linking descriptor table, index block, and flag bit block to each other:

- The index descriptor table is used to locate index block and the corresponding flag bit block.
- The bit corresponding is used to address and index table to calculate the offset in this level of index block and retrieve the serial number. Then if it's the routing table serial number is decided based on whether the corresponding flag bit is set or not.
- If the stored value in index is the serial number of sub-level index table, this serial number is taken as the array subscript of sub-level index descriptor table and the corresponding index descriptor is retrieved. The above steps are then repeated.

If compared to Figure 3, the multi-level index table is obviously more complex with more index levels and the descriptor table is added to the connection of all levels of index table. This node is added because each entry of index table (block) is changed from pointer to index.

The following issues in Figure 5 should be noted.

- For the sake of consistency of structure and code implementation, the first-level index block and the first-level flag bit block are associated to each other with index descriptor. There is only one entry in the first-level index descriptor table.
- Since the last entry of index block always stores routing table index, the flag bit array is not needed. The index descriptor table is still used to associate index block, the flag bit block pointer in descriptor table is set to "NULL" though.
- The index descriptor stores pointer of index block and flag bit block instead of array instances. Therefore, the use of index descriptor becomes more flexible as its structure can stand independent of step length of index table. Descriptors of all levels of index table may have the same structure and the number of levels and step length is customizable.

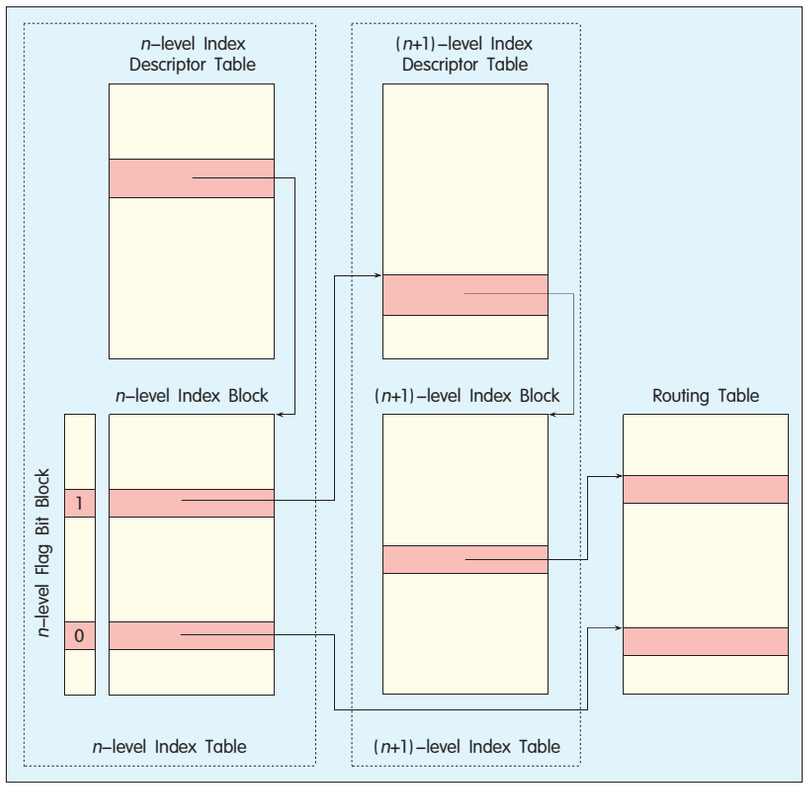
4.2 Overhead of Multi-level Index

Here is the formula for maximum memory overhead of multi-level index:

$$L=2^{x_0}+2^{x_1}M+\cdots+2\ M \quad (1)$$

$$x_0+x_1+\cdots+x_n=32 \quad (x_i<32, i<n<32) \quad (2)$$

Where L is the memory overhead of index table, M is the number of route entries, $n+1$ is the number of route levels, and $x_0, x_1 \cdots x_n$ are step lengths of all levels of index by turn. The size of the first-level index table is irrelevant to route capacity. Therefore, the memory overhead of the first-level index table is related to the step length x_0 of first-level index only and there are 2^{x_0} entries of index. Memory overhead of the second and third levels (and more) are related to route capacity and in the worst case, the number of all levels of index tables (except the first level) is the routing table capacity M. Therefore, L means the maximum number of index entries in the index table that the index table needs. The issue of smallest maximum memory overhead L boils down approximately to the problem of obtaining the extreme conditional value from the above formula. We then come to:



▲Figure 5. Multi-level index correlation.

▼Table 1. Maximum memory overhead of multi-level index

Number of Index Levels and Index Length	Maximum Memory Overhead under Different Route Table Capacities/MB						
	1K	2K	4K	8K	16K	32K	64K
20,6,6	2.3	2.7	3.1	4.2	6.4	10.8	19.5
18,7,7	1.0	1.5	2.5	4.5	8.5	16.5	32.5
20,4,4,4	2.1	2.3	2.5	3.0	5.1	6.1	8.3
17,5,5,5	0.5	0.7	1.0	2.0	3.8	7.4	14.5
16,4,4,4,4	0.3	0.5	0.8	1.5	2.9	5.6	11.1

▼Table 2. Maximum memory overhead of multi-level index

Number of Index Levels and Index Length	Maximum Memory Overhead under Different Route Table Capacities/MB						
	1K	2K	4K	8K	16K	32K	64K
16,16	257	513	1 025	2 049	/	/	/
20,12	24	40	72	136	264	520	1 032
24,8	33	34	36	40	48	64	96

▼Table 3. Number of clock cycles needed by index table lookup

Number of Index Levels and Index Length	Number of Clock Cycles Needed for Optimal Match under Different Number of Route Entries					Average Cycle	Average Time/μs
	100	200	400	800	1 600		
20,12	298	203	220	248	235	241	1.20
20,6,6	299	215	245	253	248	252	1.26
17,5,5,5	292	262	240	234	280	262	1.31
16,4,4,4,4	302	251	262	284	267	273	1.36

$$x_0=\frac{32+n\log M}{n+1} \quad (3)$$

$$x_i=\frac{32-\log M}{n+1} \quad (i=1,2\cdots n) \quad (4)$$

Where log$M$ is the routing table capacity with the base-number being 2. Transform formulas (3) and (4) and we get:

$$x_0-x_i=\log M \quad (5)$$

Formulas (1) and (2) tell us that: if routing table capacity is M, we get the smallest maximum memory overhead for the index table when step length of the first level index log$M$ is longer than any step length of the rest levels of index.

Based on formula (1), Table 1 lists the memory overhead of several typical index configurations under the condition of different numbers of route entries.

In the first column of Table 1 is index levels and index length. "20,6,6" means there are 3 levels in index table and their respective step lengths are 20, 6 and 6. The first row lists the route table capacities, that is, the maximum route entries that the routing table supports. This table tells us the maximum memory overhead in the unit of mega-bytes needed by index table under the conditions of different index lengths and different routing table capacities. Table 2 lists the maximum memory overhead corresponding to two-level index table. In case the same routing table capacity, the maximum memory overhead of two-level index table is far greater than that of multi-level index table. In other words, multi-level index table has better performance than two-level index table regarding maximum memory overhead. The "/" in Table 2 means the overhead is too large to have the necessity of counting.

We can also conclude from Table 1 that the more index levels there are, the smaller the maximum memory overhead will be. However, it's not necessarily true that we should have as more index levels as possible. This is because as the index level increases, the efficiency of routing lookup is compromised to some extent, which can be proved by the figures given in Table 3. That is to say, we should take memory overhead and lookup efficiency into account while trying to increase index levels. In an extreme case if there are 32 index levels, the multi-branch trie tree will be downgraded to binary trie tree. For more information on lookup performance of multi-branch trie tree and binary trie tree, refer to [3].

5 Conclusions

Multi-level index has more index levels and shorter step length than two-level index. Therefore, it can largely reduce the mapping space of route prefix address in the index table and thus save on memory effectively. As index adopts index serial number multiplexing, it saves half of the index storage space as compared to the pointer approach. The flag block again avoids the problem that actual memory assumption is far greater than the space required by variant, which can be caused by the approach of byte-by-byte storage and byte alignment. Multi-level index needs no index compression due to its short step length and thus makes itself less as independent on the memory management mode. Therefore, the advantage is that, the number of routing index levels and index step length can be set flexibly in accordance with the routing table capacity.

References

[1] Gupta P, Lin S, McKeown N. Routing Lookups in Hardware at Memory Access Speeds [C]// Proceedings of INFOCOM. Mar 29-Apr 2,1998, San Francisco, CA,USA. Piscataway, NJ,USA: IEEE,1998: 1240-1247.

[2] Huang Nenfu, Zhao Shiming. A Novel IP-routing Lookup Scheme and Hardware Architecture for Multigigabit Switching Routers[J]. IEEE Journal on Selected Areas in Communications, 1999,17(6): 1093-1104.

[3] Srinivasan V. Fast and Efficient Internet Lookups[D]. Washington DC,USA: Washington University, 1999.

Manuscript received: 2005-12-27

Biography

Yan Xincheng received his Master's degree from Southeast University with a major in Signal and Information System. He is a software R&D engineer engaged in the 3G IP network R&D as well as product development at the Nanjing Institute, Central Academy of ZTE Corporation.