IP-Address Lookup Using Hardware Pipelining

Pi-Chung Wang[a], Chia-Tai Chan[b] and Yaw-Chung Chen[a]
[a]Department of CSIE, National Chiao Tung University, Hsinchu, Taiwan 30050, R.O.C.
[b]Telecommunication Laboratories, Chunghwa Telecom Co. Ltd, Taipei, Taiwan
Tel: 886-3-5731851, Fax: 886-3-5727842, E-mail: pcwang@csie.nctu.edu.tw

Abstract: *Fast IP address lookup mechanism is a major design issue for the next generation routers. The existing scheme performs the IP address lookup by hardware in which the forwarding table can be compressed to fit into reasonable-size SRAM, and a lookup can be accomplished in three memory accesses. In this article, we claim that with a little extra memory, it is able to further reduce the lookup time to two memory accesses. Moreover, it can be implemented in pipelined hardware so that one routing lookup per memory access can be achieved. With state-of-the-art SRAM technology, this mechanism can achieve more than 100 million routing lookups per second.*

KEYWORD: Internet, Address Lookup, Pipelining

INTRODUCTION

Speeding up the packet forwarding in the Internet backbone requires high-speed transmission links and high performance routers. The transmission technology keeps evolving and provision of gigabit fiber links is readily available. Consequently, the key to increase the capacity of the Internet lies on fast routers [7]. A multi-gigabit router must have enough internal bandwidth to switch packets between its interfaces at multi-gigabit rates, and enough packet processing power to forward multiple millions of packets per second (MPPS). Switching in the router has been studied extensively in the ATM community and solutions for fast packet processing are commercially available. As a result, the major remaining bottleneck for the Internet router design is the slow, software-based IP lookup procedures. It is inappropriate to accomplish IP lookup for the next generation routers through software, further investigation to resolve the issue is an important and highly challenged task.

An IP lookup scheme is the most fundamental operation in any IP routing product. A packet carries a specific IP Destination Address (DA), which is a unique 32-bit field in current IP version 4 implementations. A router must search forwarding tables using a DA as the key, and determine which table entry represents the best route to forward the packet to its destination. Since the development of CIDR in 1993 [8], IP routes have been identified by a <routing prefix, prefix length> pair, where the prefix length ranges from 1 to 32 bits. For every incoming packet, a search in the router's forwarding table must be performed to determine which next hop the packet should be destined for. With CIDR, the search may be decomposed into two steps. First, it needs to find the set of routes with prefixes matching that of the incoming destination IP address. Then, among this set of routes, it has to select the one with the longest matched prefix. This is the route used to identify the next hop. Due to the facts that table entries have variable lengths and that multiple en-

tries may represent the valid routes to the same destination, the search may be complicated.

Since IPv6 is still some way off, IPv4 will stay for a while. Thus, we focus on the hardware scheme optimized for IPv4 routing lookups. This work presents a fast longest prefix matching scheme for IP switch routers. Based on our proposed scheme, the forwarding table would be reduced to enough small to fit into SRAM with very low cost. Most of the address lookups can be accomplished in a single memory access. In the worst case, the number of memory accesses for a lookup is two. When implemented in pipelined hardware, the proposed scheme can achieve one routing lookup per memory access. With state-of-the-art SRAM technology, this mechanism can achieve more than 100 million routing lookups per second. The rest of the paper is organized as follows. Section 1 describes the previous work. The proposed longest prefix matching scheme and the hardware architecture is presented in Section 2. The performance analysis of the proposed scheme is addressed in Section 3. Finally, a conclusion remark is given in Section 4.

1. RELATED WORKS

Several works have been proposed with novel data structures to reduce the complexity of longest-prefix matching lookups [2][9][11]. These data structures and their accompanying algorithms are designed primarily for software implementations, they are usually unable to complete a lookup in few memory accesses. The most straightforward way to implement a lookup scheme is to build a forwarding table for all possible IP addresses. However, the size of the forwarding table (next-hop array; NHA) is too large (2^{32} entries) to be practical.

To reduce the size of the forwarding table, an indirect lookup mechanism, as shown in Fig. 1 [3], can be employed. Each IP address is split into two parts: the segment (16-bit) and the offset (16-bit). The segment table has 64 K entries which record either the next hop of the route or pointers pointing to the associated NHAs. Each NHA consists of 64K entries, each entry records the next hop for the destination IP address. This scheme achieves a maximum of two memory accesses for a lookup in a 33-Mbyte forwarding table. By adding an intermediate-length table, the forwarding table can be reduced to 9 Mbytes; however, the maximum number of memory accesses for a lookup would be increased to three. When implemented in hardware pipeline, the scheme can accomplish one route lookup every memory access and achieve up to 20 million lookups per second.

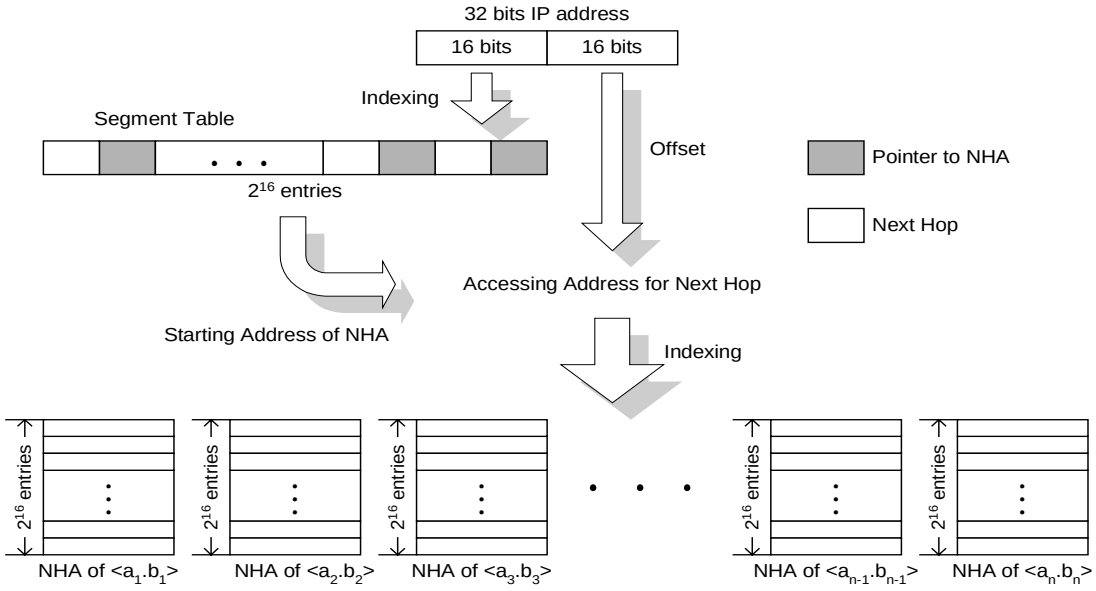

Figure 1. Indirect-lookup mechanism.

In [4], Huang *et al.* further reduced the size of the associated NHA by considering the distribution of the prefixes belonging to the same segment, as demonstrated in Fig. 2. With 53,000 routing prefixes, it results in about 1.2 Million entries in the Next Hop Array (NHA). With 1 byte per entry, the total required memory size (including segment table) is more than 1.5 Mbytes, and the required memory accesses are two. By using the compression, the required memory size can be reduced to 500 Kbytes, but the number of memory accesses is increased to three. The time complexity for building the so-called Code Word Array (CWA) and the compressed NHA (CNHA) is $O(n\log n)$ where n denotes the number of prefixes in a segment.

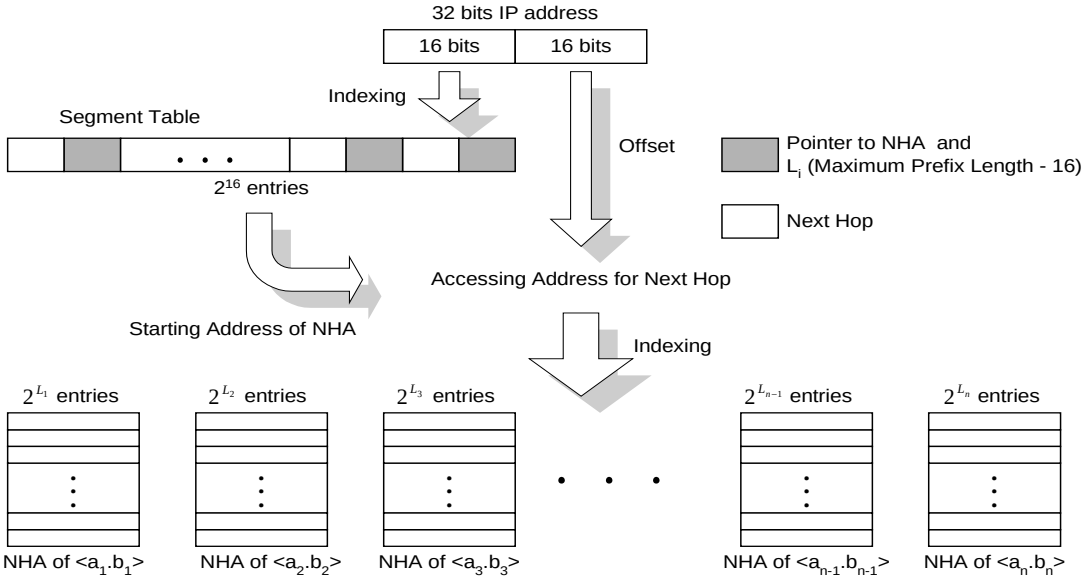

Figure 2. Indirect-lookup mechanism with variable offset length.

2. THE PROPOSED SCHEME

2.1. NHA Construction Algorithm

There are about 53,000 routing prefixes in current backbone router but it has 65,536 segments totally. In other words, there is less than one routing prefix contained in a segment on average. Moreover, we found that for most segments, there is no routing prefix to define the route, as shown in Fig. 3. Only few segments contain multiple routing prefixes, we call that routing locality: for the nearer area in the topology, the number of routing prefixes is larger, and the prefixes in the associated segment become more diverse. Oppositely, for the farther region, there are less but orderly routing prefixes within a segment. Therefore, it is possible to arrange prefixes for most segments and reduce the required memory.

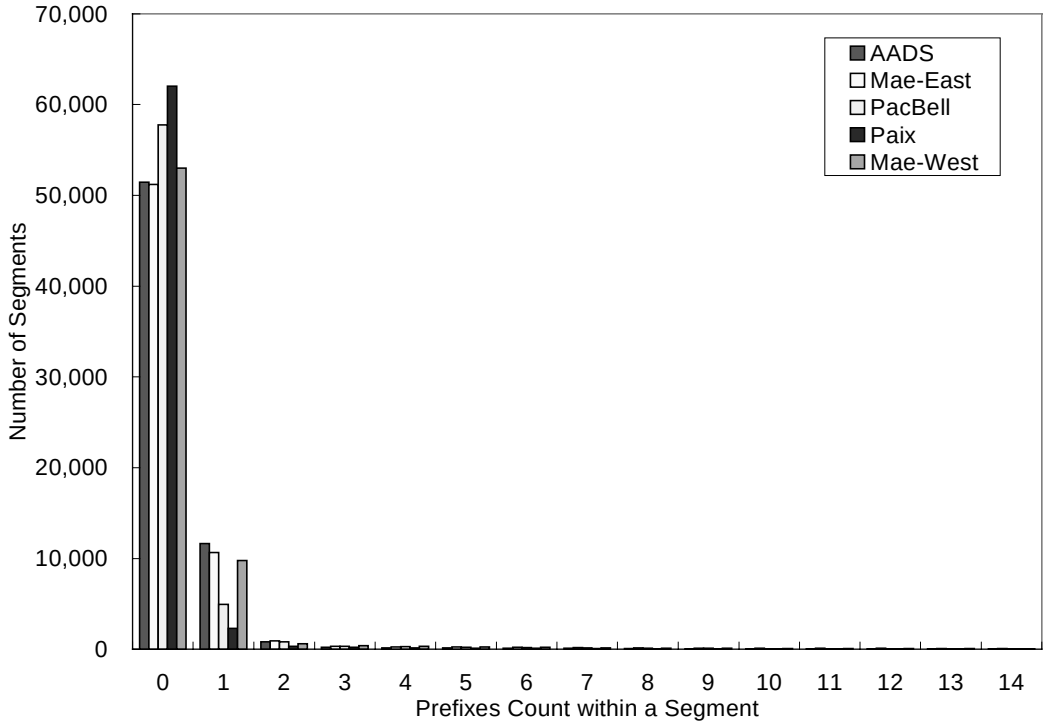

Figure 3. Prefix Count Distribution

By observing the sample routing table in Fig. 4, we can find that all routing prefixes are belonging to the same segment <63.192>. In Huang's algorithm, the total required entries in an NHA would be $2^{(20-16)}=2^4$. After a further analysis of these prefixes, we find that the first 17 bits <00111111110000000> of these prefixes are the same. Since the longest prefix length is 20, this means that we only have to record the 3-bits variation by building an NHA with $2^{(20-17)}=2^3$-entries for these 6 prefixes, as shown in Fig. 4. The longer the common part is, the shorter the NHA entries will be. The only problem is how to represent the extra prefix information. To deal with this issue, we can add few prefix bits and its length to the entry of the segment table. This will increase the size of the segment table as a trade off. Those entries with no routing prefix information, such as P_5, should be filled with a default route.

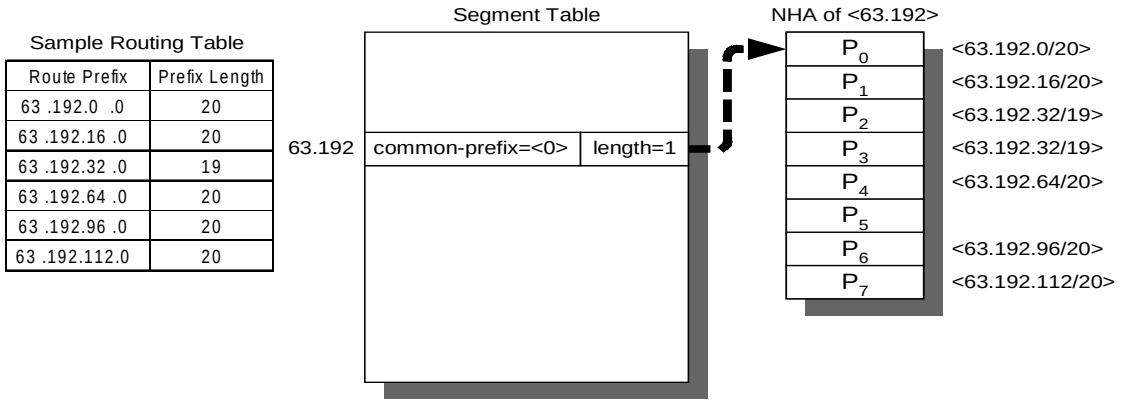

Figure 4. The resulting NHA from the sample routing prefixes.

The NHA construction algorithm for a segment is given bellow. Let l_i and h_i be the length and output port identifier of a routing prefix p_i, respectively. The *cprefix* represents the common part of routing prefixes beyond the first 16 bits in a segment, and *clength* is the valid length of *cprefix*. *Mlength* equals to the longest prefix length in the segment minus 16, thus, $clength \leqq Mlength$. Let $p_i(x, y)$ represent the bit pattern of p_i from the x_{th} bit to the y_{th} bit. The entries from $V(p_i(clength+16, l_i)) \times 2^{Mlength-(l_i-16)}$ to $V(p_i(clength+16, l_i)) \times 2^{Mlength-(l_i-16)} + (2^{Mlength-(l_i-16)} - 1)$, where $V(p_i(a,b))$ represents the value of bit pattern $p_i(a, b)$. Besides, most routers have a default route with zero prefix length which matches all addresses. The default route is used consequently if no other prefix matches. Thus the table is initially assigned the default route for possible reference to these entries.

NHA-Construction Algorithm

Input: The set of routing prefixes of a segment.

Output: The corresponding NHA of this segment.

Step 1. Let l_i and h_i be the length and output port of a routing prefix p_i, respectively.

Step 2. Let $P=\{p_0, p_1, \ldots, p_{n-1}\}$ be the set of sorted prefixes of an input segment. For any pair of prefixes p_i and p_j in the set, $i < j$ if and only if $l_i < l_j$.

Step 3. Let $cprefix=p_0(17, l_0)$, $clength=l_0$ and $Mlength=l_{n-1}-16$.

Step 4. For i=0 to n-1 do

cprefix=common bits between the $p_i(17, l_i)$ and *cprefix*.

clength=the length of *cprefix*.

Step 5. Construct the NHA table with $2^{Mlength-clength}$ entries, and set an initial value to the default route.

Step 6. For i=0 to n-1 do

Calculate the range of updated entries and set h_i.

Step 7. Stop

Let us use an example to show how this algorithm works. Consider the set of sorted prefixes in Fig. 5. In the Step 4, all prefixes are examined once for deciding the *cprefix* and *clength* value, and it results a single bit "0" for *cprefix* and thus

clength is 1. Since *Mlength* is equal to 8, the constructed NHA is with 2^{8-1}=128 entries. After constructing the NHA, the table is assigned with default route as the initial value because the routing prefixes cannot cover every entry in NHA. Then the first prefix <24.48.8/22/10> will be fetched. The 8_{th} ($=2\times2^{(8-6)}$) to 11_{th} ($=2\times2^{(8-6)}+2^{(8-6)}-1$) entries are the associated entries for <24.48.8/22/10> and will be overwritten with the output port value 10. The process is repeated for other prefixes. When processing the prefix <24.48.9/24/7>, one can find that the value of 9_{th} entry is 10, which is defined by prefix <24.48.8/22/10>, but the prefix <24.48.9/24/7> is a longer one. To satisfy the longest prefix matching, the 9_{th} entry will be overwritten with 7 again. Since the prefixes have been sorted by their lengths, this process can be done trivially.

Sample Prefixes

Prefix	Length	Output Port
24.48.8	22	10
24.48.40	22	1
24.48.56	23	10
24.48.80	23	7
24.48.82	23	7
24.48.9	24	7
24.48.10	24	10
24.48.12	24	7
24.48.13	24	1
24.48.55	24	1
24.48.84	24	7

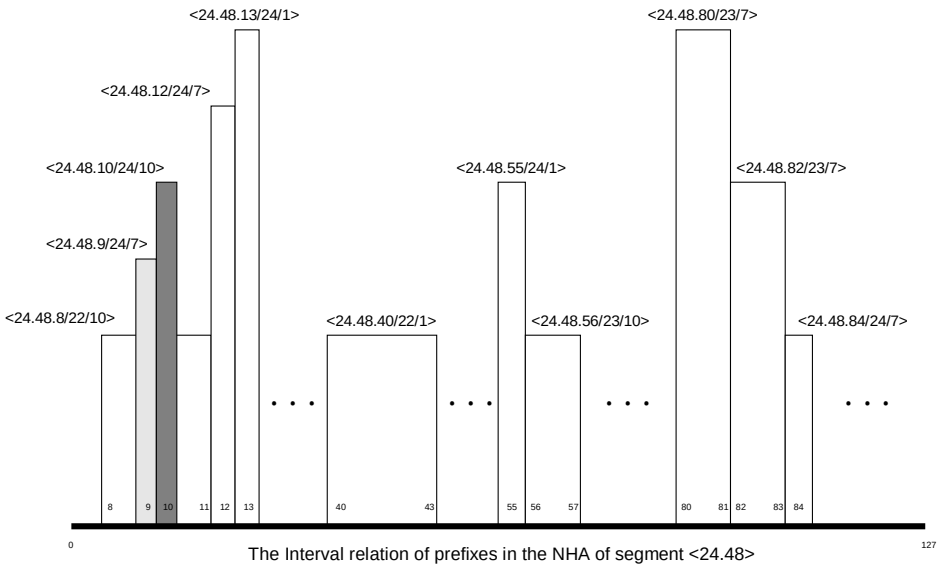

Figure 5. NHA construction example

Obviously, the computation cost is low. If a new routing prefix is received, it will recalculate *Mlength*, *cprefix*, *clength*, which is same as **Step 4**, and rebuilds the NHA if its size is changed. By applying the proposed algorithm, the number of entries of the generated NHA can be 25% less than Huang's algorithm. The detailed organization of the segment table and the hardware architecture will be addressed in **Section 2.3**. The effect of the *cprefix* is also demonstrated in **Section 3**.

2.2. NHA Compression Algorithm

In the proposed NHA construction algorithm, the size of the generated forwarding table would be around 1.2 Mbytes with one byte per entry which include 3 bits *cprefix*. Moreover, the NHA can be further compressed based on the distribution of output port count within the NHA. In Fig. 6, we show the distribution which is generated from traces of several main NAPs. We can observe that approximately 54% segments consist of less than two output ports and 96% segments use less than four output ports. Thus we could use fewer bits to encode the output port identifier in an NHA. Two extra fields, *cbits* and *base*, are appended to the entry of the segment table. At first, we use a bit-vector to record all possible output port

identifiers, which are then encoded using the smallest port identifier as the *base*. Each associated NHA entry will be set to the value of its output port identifier minus the *base*. Also, we set *cbits* to the required bits for each entry. Thus the *base* can be read before indexing the associated NHA, and the physical output port index can be calculated by adding the base index to the NHA indexing result. This process can be completed within two memory accesses plus the calculation time which can be ignored. We also provide the detailed compressed-NHA construction algorithm.

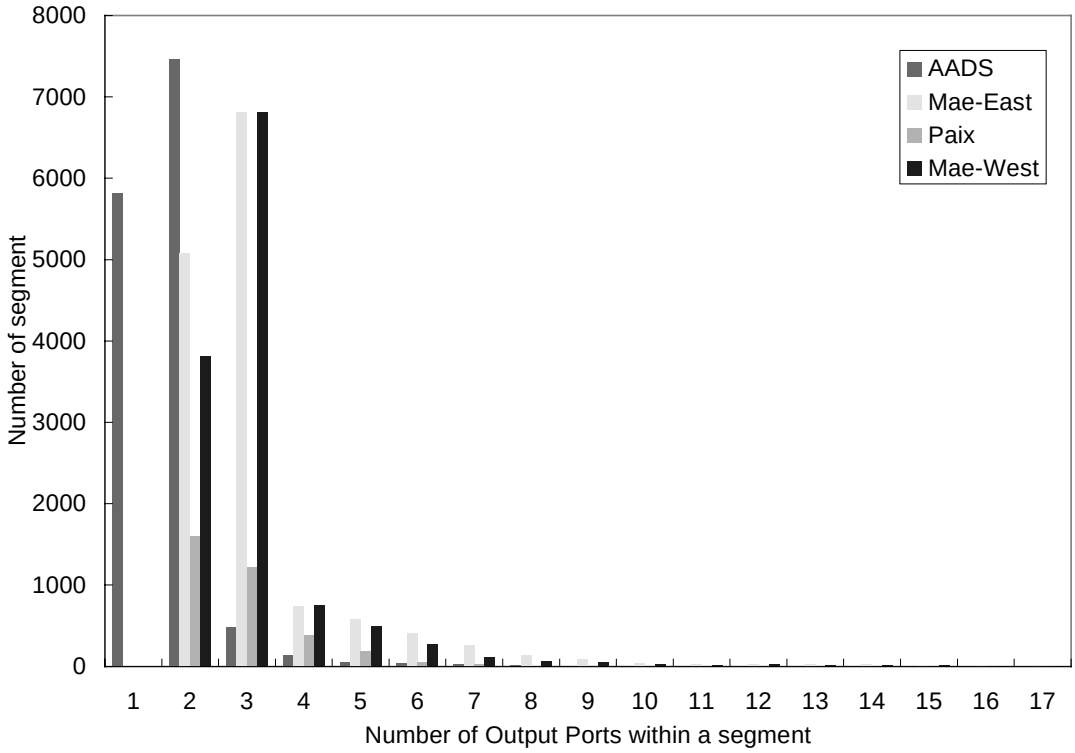

Figure 6. The distribution of Output Port Count within a segment.

Compressed-NHA Construction Algorithm

Input: The set of sorted routing prefixes of a segment.

Output: The corresponding compressed NHA of this segment.

Step 1. For each prefix, calculate *cprefix*, *Mlength* and *clength*. Also, records the output port value in the output port vector.

Step 2. Construct the index table from the output port vector and calculate the *cbits* and *base* value according to the output port count.

Step 3. Construct the NHA with size $2^{Mlength\text{-}clength} \times cbits$.

Step 4. For each routing prefix
Calculate the index value of an output port by subtracting the *base* value.
Calculate the range of updated entries in NHA and fill it with index value.

Step 5. Stop.

Although the table compression ratio in our proposed scheme is not so notable, it is able to accomplish an IP lookup with two memory accesses. Also the proposed

scheme is implementation feasible with pipelining hardware. Notice that in Huang's algorithm, the memory locations accessed in the second and the third lookup are the same because it appends the CNHA to the tail of the CWA. Therefore, it may cause the structural hazard for implementation in pipelining hardware. Such potential structural hazard can be avoided in our proposed scheme without requiring any specific hardware such as dual port memory.

2.3. Hardware Architecture

A feasible high-level hardware architecture of the proposed lookup scheme is shown in Fig. 7. When a DA is fetched, its first 16 bits are used as an index to the segment table. The value of pointer/next_hop field in the corresponding entry records the next hop if it is smaller than 256. The value will be forwarded to the selector directly. Otherwise, it records the starting address of the associated NHA. Then the bit pattern DA(*clength*, *Mlength*) is used to compare with the *cprefix* stored in this entry. If two values are identical, then *V*(DA(*clength*, *Mlength*)) plus the value of the pointer is used to access the NHA. The physical output port can be derived by adding the base with the value fetched from NHA. Otherwise, the default output port will be forwarded to the selector.

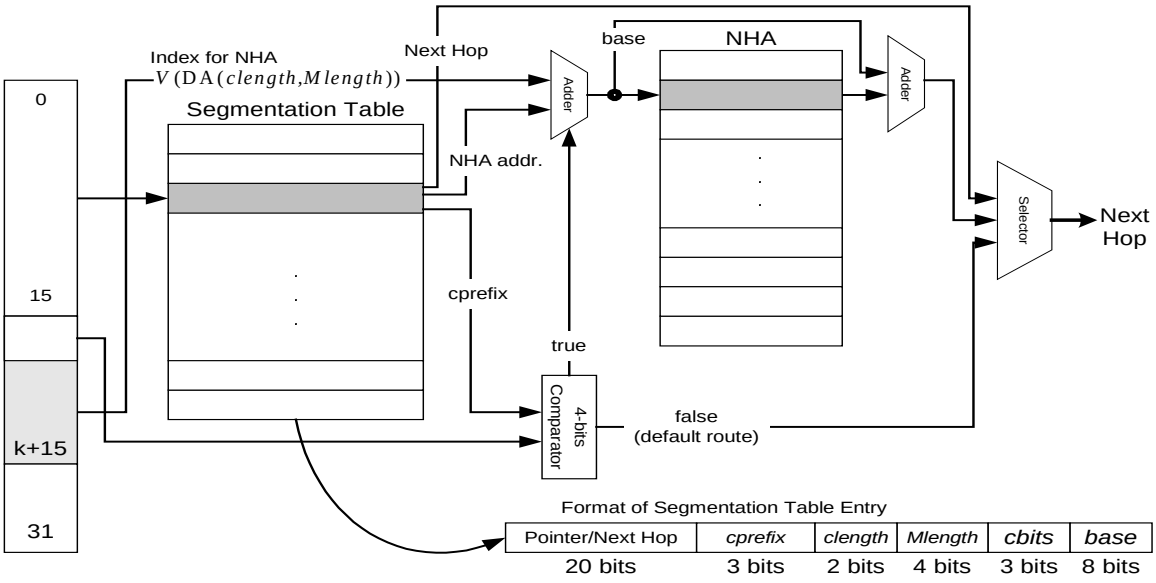

Figure 7. Hardware architecture of the proposed lookup scheme.

The entry of the segment table consists of 6 fields: pointer/next hop, *cprefix*, *clength*, *Mlength*, base and *cbits*. The length of pointer/next hop is 20 bits which can map to 1 Mega memory addresses, this is more than five times the space required for storing the whole NHAs generated from the real world routing tables. Since the maximum prefix length minus the length of segment (16) is smaller than 16, the length of *Mlength* is 4 bits. Trivially, the base is 8 bits and the maximum number of encoded bits is 8. Thus the bit count of *cbits* is 3. If we use *cprefix* with length 3, then the *clength* is 2 bits. As a consequence, the length of each entry is 40 bits, and the size of the segment table is $2^{16}\times 40$ bits, i.e., 320 Kbytes.

3. PERFORMANCE ANALYSIS

Our proposed IP address lookup scheme aims at two targets, one is to reduce the memory space needed for the forwarding tables, and the other is to speed up the IP lookup process. Through simulation, we show that the proposed scheme features a low memory requirement while achieving very high IP lookup performance. Note that the current backbone routers have a routing table with about 53,000 entries. We use the logs of publicly available routing tables as the basis for comparison. These tables are offered by the IPMA project [6], they provide a daily snap shot of the routing tables used by some major Network Access Points (NAPs).

To further realize the effect of the extra *cprefix*, we use the trace available in the router of Mae-East NAP to present the relation between the total number of entries in NHAs and the length of *cprefix*, as shown in Fig. 8. By coupling the effect of *cprefix*, the number of entries can be 25% less than that required in Huang's algorithm. Although as the length of *cprefix* increases, the number of NHA entries is reduced, it has to increase the entry length of the segment table, thus results in a larger segment table. From the observation of Fig. 8, the suitable *cprefix* length should be no more than 3 by considering both segment table size and the number of NHA entries.

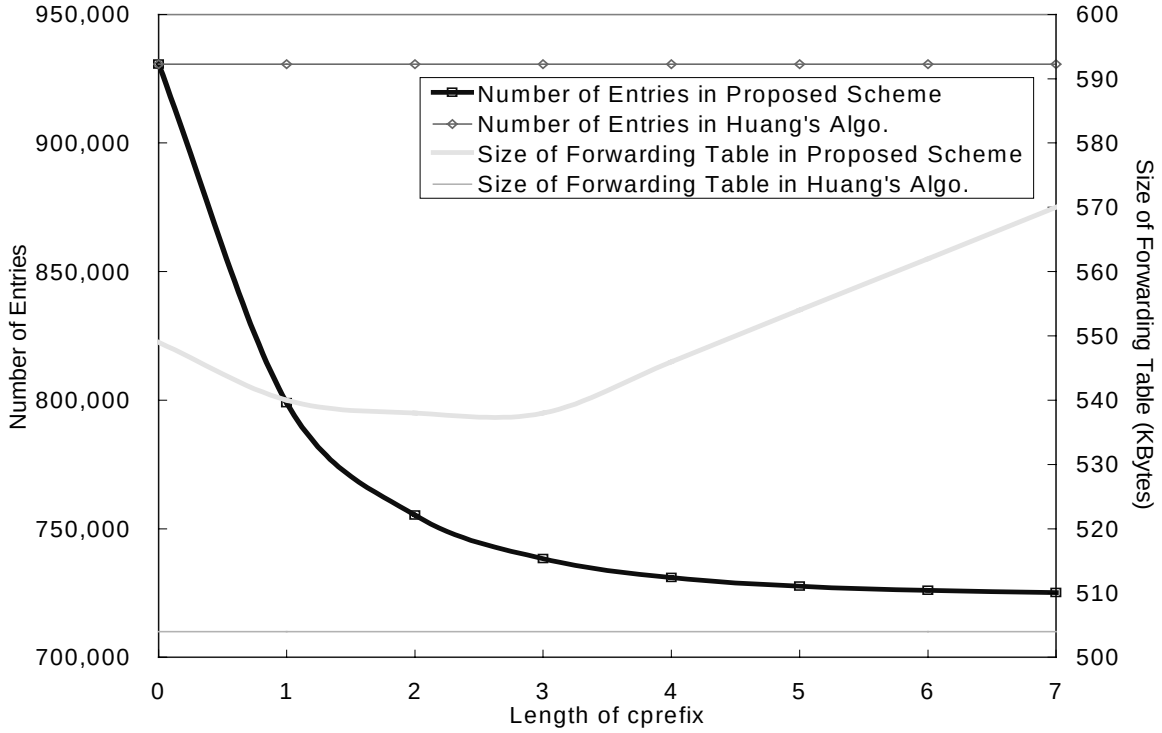

Figure 8. Effect of the Length of *cprefix*.

In Table II, we log five traces to build the forwarding table for illustrating the effect of the proposed scheme. Although the size of the forwarding table might become larger as a trade-off for throughput improvement, we can find that the required memory size does not increase too much, or even decrease in two traces (PacBell and Paix). This is because that the use of *cprefix* reduces the number of

NHA entries effectively. While in the traces of two backbone NAPs (Mae-East and Mae-West), the memory increment is notable, this is due to the diverse routing prefixes in the backbone, in which the effect of cprefix is degraded.

Site	Routing Entries	Memory Usage of Proposed Scheme (Kbytes)	Memory Usage of Huang's Algo. (Kbytes)
AADS	23,426	396	403
Mae-East	53,226	538	504
PacBell	29,942	258	412
Paix	11,498	328	351
Mae-West	33,332	529	454

Table II. The comparison of memory requirements.

4. CONCLUSION

In this work, we propose a fast IP-address lookup scheme which is implementation feasible with nowadays high-speed SRAM. Our scheme can complete a lookup within two memory accesses. From the simulation results, the required memory is no larger that 540 Kbytes. With two memory accesses for an IP lookup, the proposed scheme can avoid the structural hazard in hardware pipelining. By applying state-of-the-art SRAM (i.e., 5 ns access time) technology, our scheme is able to achieve more than a hundred million routing lookups per second.

REFERENCES

[1] S. Deering and R. Hinden: “Internet Protocol version 6 (IPv6) Specification.” RFC 1883, 1996.

[2] M. Degermark, A. Brodnik, S.Carlsson, and S. Pink: “Small Forwarding Tables for Fast Routing Lookups.” In Proc. ACM SIGCOMM ’97, p.p. 3-14.

[3] P. Gupta, S. Lin, and N. McKeown: “Routing Lookups in Hardware at Memory Access Speeds.” In Proc. IEEE INFOCOM ’98, March 1998, p.p. 1240-1247.

[4] N.F. Huang, S. M. Zhao, and J. Y. Pan: “A Fast IP Routing Lookup Scheme for Gigabit Switch Routers.” In Proc. IEEE INFOCOM ’99, New York, USA, March 1999.

[5] S. Bradner: “Next Generation Routers. Overview.” In Proc. Networld Interop 97, 1997.

[6] Merit Networks, Inc. Internet Performance Measurement and Analysis (IPMA) Statistics and Daily Reports. See *http://www.merit.edu/ipma/routing_table/*.

[7] S. Keshav and R. Sharma: “Issues and Trends in Router Design.” IEEE Commun. Mag., May 1998, vol. 36, no.5, p.p. 144-151.

[8] Y. Rekhter, T. Li: “An Architecture for IP Address Allocation with CIDR.” RFC 1518, Sept. 1993.

[9] M. Waldvogel, G. vargnese, J. Turner, and B. Plattner: “Scalable High Speed IP Routing Lookups.” In Proc. ACM SIGCOMM ’97, p.p. 25-36.

[10] B. Lampson, V. Srinivasan and G. Varghese: “IP Lookups Using Multiway and Multicolumn Search,” IEEE/ACM Trans. On Networking, June 1999, vol. 7 no. 4, p.p. 324-334.

[11] S. Nilsson and G. Karlsson: “IP-Address Lookup Using LC-Tries,” IEEE JSAC, June 1999, vol. 17, no. 6, p.p. 1083-1029.