Scalable IPv6 Lookup/Update Design for High-Throughput Routers

Chung-Ho Chen, Chao-Hsien Hsu, Chen-Chieh Wang
Department of Electrical Engineering and Institute of Computer and Communication Engineering
National Cheng-Kung University
Taiwan, R.O.C.
chchen@mail.ncku.edu.tw, hsien@casmail.ee.ncku.edu.tw, ccwang@mail.ee.ncku.edu.tw

Abstract

Achieving scalable performance in the IPv6 address lookup and update poses a challenge to the design of existing routers. To concurrently match address prefixes with different route entries, we propose a parallel memory lookup scheme which uses three-level tables to cover various lengths of prefix distributions for the long IP address. The scheme employs a parallel CRC address compression hardware to reduce the lookup table sizes. The multi-cycle implementation of the design has achieved an average of 1.6 memory accesses per lookup request. The pipeline version features a five-stage pipeline design with a mechanism to reduce pipeline stalls due to updates. Performance simulation reveals that the number of address queue entries significantly influences the lookup throughput when frequent table updates occur. The proposed single pipeline module with an eight-entry queue stage has achieved a maximum rate of 100×10^6 lookups per second. With the four-pipeline configuration, the throughput is increased by a factor of 2.5 for sparse updates and up to 2.3 when the update rate increases to 20 percents of the lookup's. This paper has demonstrated a viable IPv6 lookup design that is scalable for high-throughput routers.

Keywords: IPv6 lookup, longest prefix match, route update, scalable throughput.

1 Introduction

The longer address length and larger address space in IPv6 pose a challenge to the design of IP address lookup. The address lookup involves the longest prefix match (LPM) operation, i.e., finding an entry in the lookup table with the longest prefix that matches with the incoming packet's destination IP address. The longer IPv6 address lengthens the latency of an LPM operation and thus slows down the lookup rate in routers. For instance, with an M-trie based search, the number of memory accesses required equals to the depth of the tree. To lookup an IPv6 address, up to eight accesses are required, more than doubling the latency of lookup in IPv4 [1].

Extending IPv4-based schemes in IPv6 may also suffer from performance degradation due to the different prefix length distributions in IPv6 routing tables [2]. The global IPv6 unicast address is partitioned into several segments as a hierarchical tree such as ISP, Site or LAN [3]. Observing the address allocation policy and the format of the aggregatable global unicast addresses, it is known that the prefix length mainly distributes among the TLA (Top Level Aggregator), NLA (Next-Level Aggregation), or SLA (Site-Level Aggregation) fields. Apparently, the prefix length distribution for a switching router changes with the hierarchical level at which the switching router is used.

An address lookup scheme must be scalable in terms of memory usage and more importantly achieve scalable lookup and update performance. Nonetheless, the lookup throughput is hindered by the update operations due to route changes, which reportedly changes at a rate exceeding several hundred prefix updates per second [4]. To tackle the above problems, first performing concurrent lookups in multi-range prefixes, we propose a parallel memory lookup scheme with reduced table sizes to cover various lengths of prefix distributions for the long IP address. The multi-cycle implementation of this design has achieved an average of 1.6 memory accesses per lookup request.

Next, we present the pipeline design of this lookup scheme considering the update operations which are quantified by the rate of the occurrences and the latency. This five-stage pipeline design includes a queue stage that buffers the lookup requests while updates are being performed. We evaluate the effect of the queue entry number and the multiple-pipeline configuration on the lookup throughput. The performance evaluation shows that the proposed design is scalable both to the number of lookups and update requests for the IPv6 address lookup system.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 presents the proposed address lookup scheme with examples. Section 4 describes the implementation in multi-cycle as well as in pipeline design. Section 5 discusses the simulation results of the proposed scheme. Finally, the conclusion is given in Section 6.

2 Related Work

An extensive survey on IP address lookup algorithms can be found in [4] with the focus on trie-based schemes which are also used in [5-9]. In [5], the IP address lookup problem is modeled as a searching problem on a binary-trie which is partitioned into 4-level of subtrees for pipeline implementation. Similarly based on the trie structure and prefix partition, Chang et al. proposed a lookup table design for IPv4 system [6]. The authors in [8] proposed a modified PATRICIA trie with the help of hash tables to speedup the search. In [10], the core method treats each prefix as a range and encodes it using the start and end of the range. Then the range entries are arranged in a binary search table with a mapping established between the table and the corresponding prefix. Differently, the approach in [11] uses binary search over the hash tables organized by the length of the prefix.

To use the IP address as a memory pointer for table lookup, a hashing function that takes the longer address is often used to produce a shorter index so that the required memory size can be reduced [12-13]. In [13], the lookup architecture uses parallel EXOR hashing logic for each single prefix address table and provides mechanisms for collided mapping. For IPv4, the scheme requires one to five memory accesses with a small memory requirement. In contrast, content addressable memories (CAMs) match the incoming IP addresses directly with the contents [14-17]. In [14], the routing lookup method based on bi-search on prefix length is proposed. It is implemented in a pipeline structure taking the advantages of the traditional ternary content addressable memories (TCAMs). However, for a larger forwarding table, using TCAM-based lookup can be very expensive in terms of memory bits used and power dissipation. TCAMs are expensive because every route bit needs two SRAM cells. As a result, TCAMs are the choice for applications that require the ability to mask certain bits in each entry to enforce policy lookups or access control rules [16]. For cost effectiveness, policy lookups and forwarding lookups are often separately implemented with different hardware systems [1].

In [18], a hardware scheme that improves lookup memory access in IP lookup is proposed for IPv4. This lookup architecture uses a table storing all route prefixes that are up to 24-bits long and a second table for prefixes that are longer than 24-bits. Our scheme differs from theirs in many aspects. First, to be used in IPv6, we propose the use of a parallel CRC technique to reduce table sizes. Second, the arrangement of the parallel tables in the hierarchy reduces the number of memory accesses, which in turn makes fast lookup possible for the long prefixes in IPv6. More importantly, we present the scalable pipeline design which is not previously unveiled.

3 Longest Prefix Matching with Parallel Memory Lookup

To design a lookup scheme tailoring directly for pipeline operation, we propose to do longest prefix matching with a parallel memory lookup (PML) scheme that consists of a three-level memory hierarchy. The organization of the scheme is shown in Figure 1 where the NLA ID and SLA ID are partitioned into several 8-bits long segments.

The scheme organizes the lookup tables in a hierarchical layout. First, the first-level table is the TLA-table (TLAT) which has 2^{13} entries storing all the possible route prefixes that belong to the TLA field. Then, three second-level tables are used for the different ranges of prefix length. The SLT40 table stores the prefixes of length that is greater than 24 bits and up to 40 bits. The other two tables, SLT48, and SLT56 store all the route prefixes that are equal to 48 bits and 56 bits respectively. The rest of the route prefixes

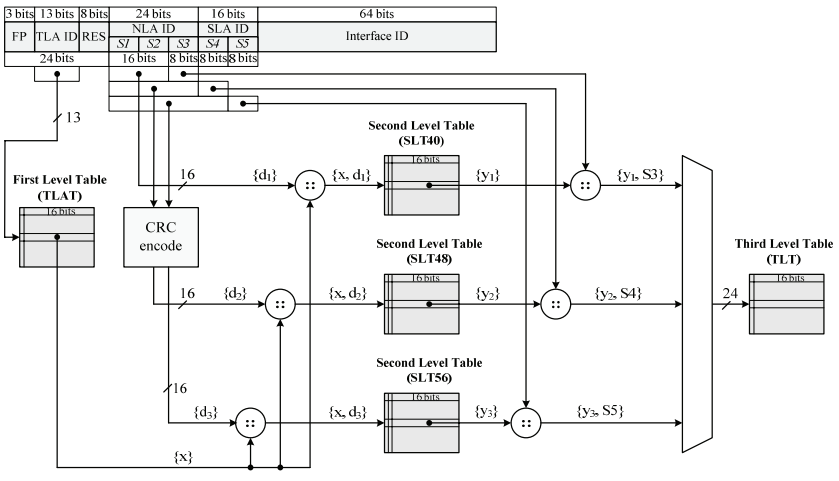


Figure 1 The proposed parallel memory lookup (PML) scheme (:: denotes concatenation)

(prefix length greater than 40 bits and less than 48 bits, greater than 48 bits and less than 56 bits, and greater than 56 bits and up to 64 bits) are stored in the third-level table (TLT). The reason to layout lookup tables in this way is to put the most likely prefix distributions in the second-level tables for parallel accesses. The specifications of the table entries are shown in Figure 2.

3.1 Address Lookup and Entry Insertion

When an IPv6 destination address is presented to the address lookup scheme, the following steps are taken:

I. Use the TLA field of the destination IP address as the index to access the TLAT table. The result is either the base address (x) for the second-level tables or the output port identifier. It depends on the number of TLA ID to decide the bit length of (x). In the simulation result, the bit length of the base address (x) is 8.

II. For the former, the base address is concatenated with the following numbers respectively: (1) with the number from {*S1*, *S2*}, (2) with the CRC output that encodes {*S1*, *S2*, *S3*}, and (3) with the CRC output that encodes {*S1*, *S2*, *S3*, *S4*}. The three resultant addresses are used in parallel to access the SLT40, SLT48, and SLT56 respectively. The outcome is either the longest prefix match found in the second-level tables or the base address (y) for the third-level table.

III. For the later, depending on where the base address is obtained (from SLT40, SLT48, or SLT56), it is concatenated with the respective segment ({*S3*}, {*S4*}, or {*S5*}).

IV. Last, the output port is found in the TLT table.

Suppose that a prefix *P* is to be inserted into the table. If *P* conforms to the length of the TLA field, its output port identifier is stored in the TLAT table. Otherwise, the addressed entry is written with the assigned base address that is used for accessing the second level tables for parallel lookup. The SLT40 table contains all route prefixes that are greater than 24 bits and up to 40 bits. Prefix length in this range is allocated $2^{40-prefix\_length}$ entries in the SLT40 table. This is because the number of the don't-care bits is *40 - prefix length*, considering the way SLT40 is addressed. When the prefix length is 40 bits, only one entry (2^0) is allocated for it in STL40. Consequently, an entry corresponding to one of the $2^{40-prefix\_length}$ entries in SLT40 shares the single 24-bit prefix in the TLAT table. For example, the route prefix 20:01:00:13/32 is allocated 256 entries (2^{40-32}) in SLT40, ranging from **01**:13:00 to **01**:13:FF (concatenating the base-address **01** obtained from TLAT with 13:00 which comes from segment {*S1*, *S2*} in the route prefix).

If the prefix length is greater than 40 bits and less than 48 bits, it is stored in the TLT table and there are $2^{48-prefix\_length}$ entries associated with the route prefix in TLT. In this case, the addressed entry in SLT40 contains the base address for these entries in TLT. For a prefix length that is greater than 48 bits and less than 56 bits, $2^{56-prefix\_length}$ entries are allocated in the TLT. The base address is obtained from SLT48. To access the SLT48 table, the address is obtained by concatenating the base address from TLAT with the CRC output that encodes {*S1*, *S2*, *S3*}. Similarly, if the prefix length is longer than 56 bits and up to 64 bits, there are $2^{64-prefix\_length}$ entries allocated in the TLT table. In this case, the base address for the TLT table is obtained from the SLT56 table. The parallel CRC encoder uses $X^{16} + X^{12} + X^4 + X^1 + 1$ as the polynomial.

3.2 Lookup Table Initialization and Update

In this section, we present the process for filling up the lookup tables based on a list of given prefixes. The example shows how a control processor establishes the data structure for the given prefixes, initializes the lookup tables, and maintains the data structure for the insertion and deletion of entries in the lookup tables.

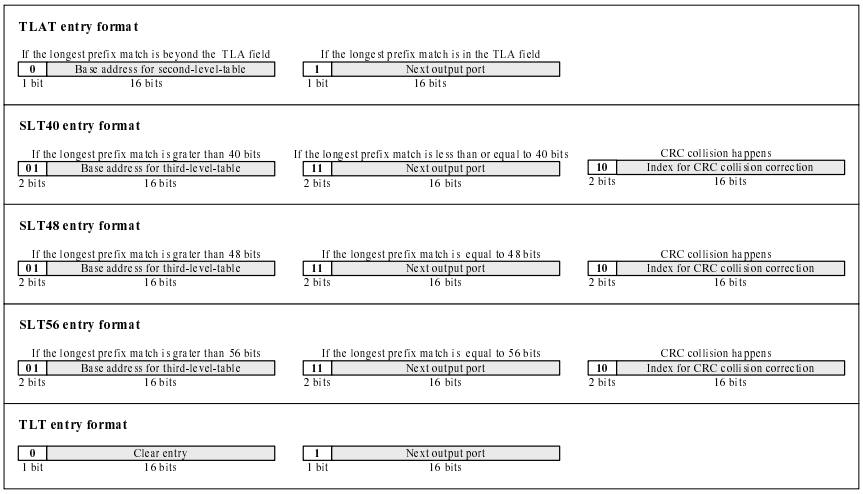


Figure 2 Entry format of the lookup tables

The preprocess function run by the control processor constructs the linked-list data structure that is used to initialize and update the lookup tables when a route change occurs. To do this, the control processor issues the update commands to the interface unit of the lookup engine. Each node in the linked-list has four items: *prefix64*, *length*, *limit*, and *hop*. For instance, if prefix Y is 20:12:00:01:E4:5F:70/52, then the *prefix64* field is 20:12:00:01:E4:5F:70:00, padding 0 to 64 bits. The *length* field is simply the prefix length. The *limit* field represents the number of entries to be allocated in the chosen table, in this case 2^{56-52} entries in TLT, and the *hop* field is the output port identifier.

For longest prefix matching, the preprocess function checks if the address range of an input prefix overlaps with the existing one in the lookup table. For a received route prefix whose address range is not overlapped with existent prefixes in the linked list, a new node is directly constructed for the input prefix. Otherwise, the preprocess function checks whether the input prefix and the overlapped prefix belong to the same second-level table. If they are in the same second-level table, the preprocess function splits the overlapped address space at the boundary for longest prefix matching. If the two overlapped prefixes are not accessed from the same second-level table, the preprocess function relocates the overlapped blocks of the shorter prefix in the TLT table to allow the access from the second-level table assigned according to the longer prefix.

As an example, assume that there are six prefixes: A, B, C, D, E, and F shown in Figure 3 to be prepared for the lookup table. Initially, a new node for prefix A is constructed and inserted into the linked list. Since the address ranges of prefix A, B, C, and D are not overlapped with each other, prefix B, C, and D are inserted into the linked list with the same procedure used by prefix A. For the time being, the resultant linked list is shown in Figure 3. The process proceeds with prefix E which is overlapped with prefix D. Since their lengths are greater than 48 bits and less than 56 bits, they are accessed from the same second-level table, SLT48. For longest prefix match, the overlapped address space from 20:12:00:01:E4:5F:70: to 20:12:00:01:E4:5F:7F: is now assigned to prefix E and consequently prefix D is revised. The number of entries allocated to E in TLT is 16 (2^{56-52}). Figure 4 illustrates the resultant data structure.

Because of its prefix length, the last prefix F, which is overlapped with prefix D, is accessed from SLT56 different from the shorter prefix D. To include prefix F in the lookup table, the 2^8 entries out of prefix D that covers prefix F are made accessible from SLT56 instead of the SLT48 table. The complete lookup table is now shown in Figure 5 in which the dotted blocks are the moved entries for prefix D.

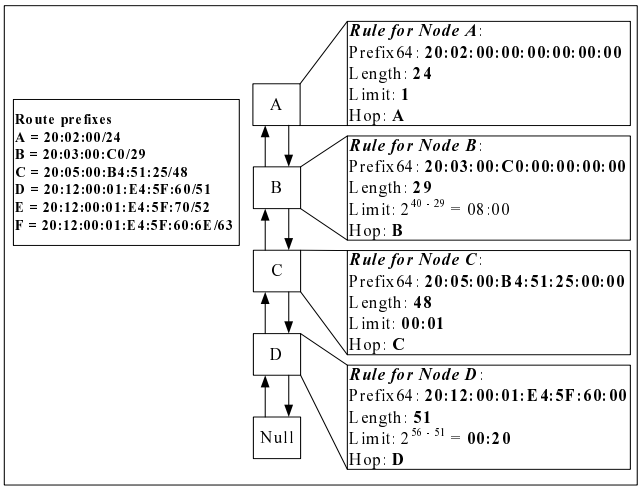


Figure 3 The linked list after prefix A, B, C, and D are inserted

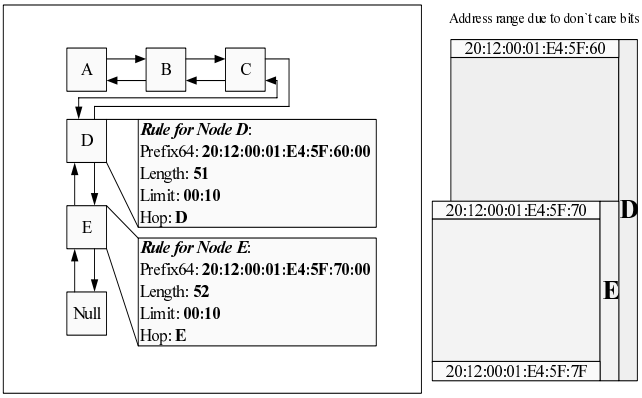


Figure 4 Prefix E is overlapped with the prefix D; both of which are accessed via the same second-level table: SLT48

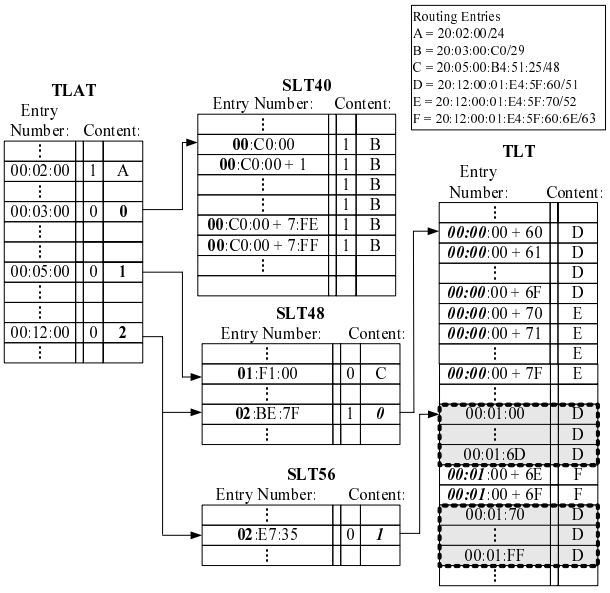


Figure 5 An example shows the insertion of routes in the lookup tables for the given prefixes

4 System Implementation

First, we present the implementation of the lookup engine in a multi-cycle design. Figure 6 shows the multi-cycle datapath and the finite state machine (FSM) for the design. The lookup FSM transfers to S1 state as the incoming packet is received. In S1 state, the TLAT table is accessed. If no LPM is found, the control transits to S2 where all the second-level tables are accessed in the same cycle. Similarly, if no LPM is found, the state transits to S3 for the lookup in the third-level table. During the lookup in the second-level table, if a CRC collision is found from the addressed entry, the state transfers to S4 to activate the correction operation that directs the access to the small table inside the unit. This table maintains the mapping between the collided address and its next hop identification.

4.1 Pipeline Design

The development of the pipeline micro-architecture for the PML scheme is based on the lookup procedure depicted in Figure 6. The PML pipeline consists of five stages, which are TLAT lookup, address queue, second lookup, third lookup, and write back. Figure 7 illustrates the pipeline architecture.

At the TLAT lookup stage, the TLA field of the incoming destination address issued by the lookup/update dispatch unit is used to consult the TLAT table. The output result, which is either the next hop or the base address, of the first stage is then written into the queue stage. The address queue is a circular queue which buffers lookup requests in case of stalling due to the on-going update process. Each queue entry has three fields for the status, incoming destination address, and the output result from TLAT. The status field includes the following states: *clear* (empty queue entry), *wait* (waiting for the result of address lookup), *finish* (destination address lookup in the queue entry is finished), and *block* (the entry is in block state before the update of the lookup table is completed). The allocation of the address queue is governed by the head-index and tail-index pointers. If there is a new destination address received from the previous stage, the new address is stored into the corresponding entry pointed to by the tail-index pointer and the status is set as *wait*. The entry pointed to by the head-index is chosen for address lookup. When the blocked entries are released, they have a higher priority than the other entries to enter the next stage.

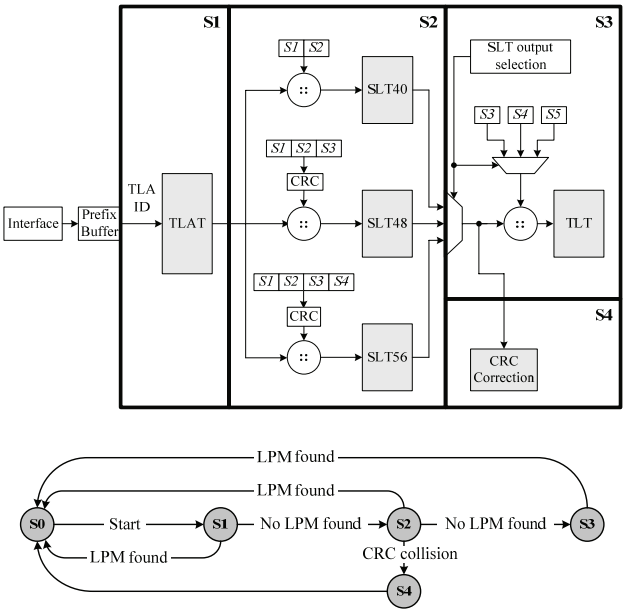


Figure 6 Multi-cycle implementation and its finite state machine

The third stage of pipeline is the Second Lookup stage where all the second-level tables are accessed simultaneously by using the respective concatenated addresses. If there is a CRC collision at the third stage, the collision entry is bypassed to the next stage.

The fourth stage is the Third Lookup stage that uses the pointer from the previous stage to access the third-level table or performs the CRC correction that is the same as the operation in the S4 of multi-cycle implementation. Finally, the write back stage writes back the lookup result into the LPM match register and releases the entry in the address queue.

4.2 Lookup and Update Coordination

Updating (insertion or deletion) entries for a prefix in the lookup table takes a similar action depicted in Figure 6. For the pipeline, the update request is also issued by the lookup/update dispatch unit. For an update request, depending on the prefix, the corresponding lookup tables are updated accordingly. While the entries of the second-level and/or the third-level tables are being updated, new lookup requests can be queued in the queue stage. The circular queue at the second stage of the pipeline is used to mitigate the performance loss due to route update. The circular storage can buffer the lookup requests blocked by

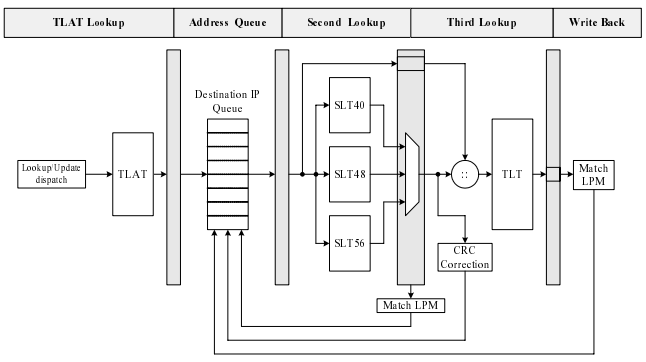


Figure 7 The pipeline architecture supporting IP address lookup and update

Table 1 Prefix length distributions

	A (%)	B (%)	C (%)
24-40	5	10	34
41-48	15	30	57
49-56	20	40	6
56-64	60	20	3
total	100	100	100

the update procedure. The number of the circular queue entries is a factor that determines the lookup performance as will be shown later.

4.3 Multiple-Pipeline Configuration

To improve the throughput of address lookup, a system employing multiple pipelines can be used. In this case, the dispatch interface dispatches the incoming route prefix to the addressed pipeline according to the most significant bits of the TLA ID field. To configure this system, the preprocess function assigns the incoming route prefix to the addressed pipeline. Thus, in the linked list, a field called *pipeline* is added into the node to indicate which pipeline the incoming route prefix belongs to. The preprocess function classifies all the route prefixes by the TLA field of the destination address and assigns each route prefix to the addressed pipeline module.

5 Simulation System and Results

To evaluate the performance of the proposed scheme, we refer to the use of a synthesis IPv6 routing table [2] which inherits the features of IPv4 tables. Three test patterns A, B and C are generated and their prefix length distributions are shown in Table 1. Test pattern C is similar to the prefix length distribution from Figure 6 of [2] where /48 prefixes dominate. Test pattern A and B are generated for the comparisons of the performance.

Our simulation system consists of the following parts. First, the random route prefix generator generates the route prefixes. Second, the route prefixes are fed into the preprocess function run by the ARM ADS tool kits. The preprocess function constructs the linked list corresponding to the route prefixes. Finally, the linked list is used as the test bench to initialize the lookup tables in the lookup engine which is implemented in Verilog and simulated in the ModelSim environment that generates the simulation results. Using the TSMC 0.25u technology, the achieved clock frequency for the PML pipeline engine is 100 MHz. The critical path of the pipeline is the queue stage assuming the memory for the lookup tables are available.

5.1 Results and Discussions

To determine the output length of the CRC module to be used, we evaluate the number of CRC collisions per 10^6 memory entries. The simulation results are shown in Figure 8. Test pattern C has fewer collisions per 10^6 memory entries than test pattern A and B. For test pattern A and B, about one collision occurs per 10^6 memory entries when the length of CRC output is set to 13 bits. As the CRC output is increased to 16 bits, the CRC collision virtually ceases to occur. For this reason, this design proposes the use of a 16-bit CRC output for the PML lookup scheme.

The worst case occurs when a lookup needs to access the TLT. Thus, it always takes three memory accesses to finish address lookup in the multi-cycle implementation. For the pipeline implementation, the increased number of memory accesses does not impact the throughput.

5.1.1 Number of Memory Accesses in Multi-Cycle Architecture

For the multi-cycle architecture, the proposed scheme takes one to three memory accesses to complete the address lookup. The average number of memory accesses required is shown in Figure 9. For test pattern C, with less than two memory accesses on the average, the output port can be identified. In addition, we also compare the PML scheme with the previous work, DP-Trie [9], MultiWay [10], and Binary Search [11]. Achieving an average of 1.6 memory accesses in test pattern C, the PML design has outperformed the rest in comparison. This result comes from the arrangement of the lookup tables that enable simultaneous accesses for the different ranges of prefixes.

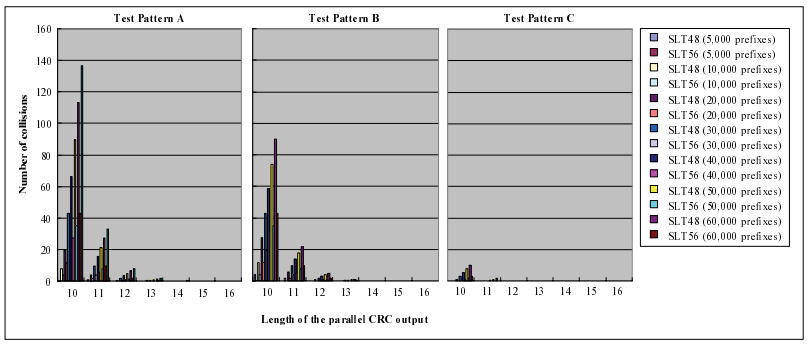


Figure 8 Collisions per 10^6 memory entries

5.1.2 Memory Requirement

The memory requirement of the proposed scheme is evaluated and the result is shown in Figure 10. The total occupied entries take about 20 MB to 26 MB of the memory for 60,000 prefixes, roughly the same size with respect to other schemes except the DP-Trie scheme. Noticing the way the table address is concatenated, the most significant bits (x or y) are assigned by the preprocess function and thus one can allocate the memory entries linearly from the lowest address for each table. The utilization of each lookup table depends on the distribution of the prefix length, which changes with the hierarchical level where the switching router is used. Also it depends on the allocation of the prefix lengths the table covers. In general, the SLT40 and TLT table require a larger size because the prefix lengths they cover. To adapt to the variation of prefix distributions, a programmable technique can be used to configure the memory size for each lookup table. Consequently, the required memory space of our scheme does not increase significantly as the route prefixes is increased due to the use of the parallel CRC compression method.

5.1.3 Pipeline Throughput

For the lookup pipeline, we present the pipeline throughput, which is defined as the number of address lookups per clock cycle. The simulation results for single pipeline and four-pipeline systems are shown in Figure 11. An update operation clearly stalls the address lookup process in the pipeline. The number of the queue entries at the second stage is a significant factor that affects the throughput of address lookup.

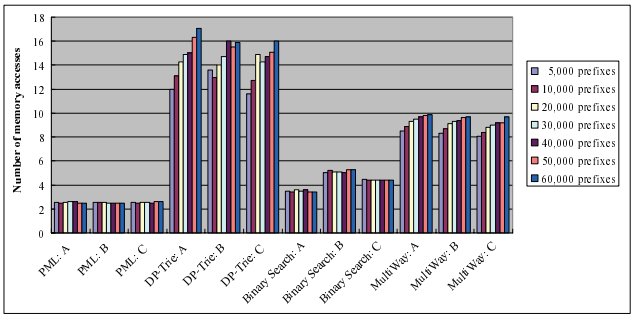


Figure 9 Comparisons in the number of memory accesses

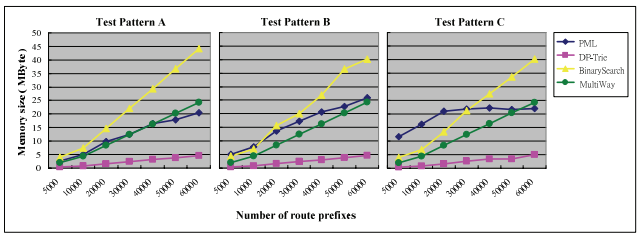


Figure 10 Comparisons of memory requirement

We examine different entry number in the circular queue to observe the variation of the throughput. The update operations are represented by how many address lookups are stalled due to the update requests. For instance, when the percentage of the update operation is 20%, this indicates that for every five address lookups, there is one update request. In this simulation, there are 60,000 prefixes of test pattern C and the latency of an update is evenly distributed between one to 2^{16} cycles; the latter is the worst case for updating the SLT40 table.

For a four-entry circular queue, the throughput degrades rapidly. Examining the throughputs obtained, an eight-entry circular queue appears to be the cost-effective configuration. When updates are rare, the pipeline throughput is close to one for the single pipeline system. In the multiple-pipeline architecture where four pipeline modules are used, the design achieves a throughput of about 2.5 address lookups per clock cycle when updates are rare. The loss of throughput in this multiple-pipeline configuration comes from the fact that not every pipeline is used at the same time for the input steam.

An interesting result comes up, which reveals that the impact of updates on the throughput for the multiple-pipeline configuration is decreased. This can be explained as follows. With multiple-pipelines, if some pipeline modules are halted by the update operations, others can continuously perform the lookup requests. Thus, the multiple-pipeline architecture not only extends the lookup ranges for the routing table but also reduces the impact of route updates.

6 Conclusion

In this paper, we propose an address lookup/update scheme for IPv6 system. We propose the use of three-level tables to cover various lengths of prefix distributions for the long IP address. To achieve concurrent lookups of different prefix lengths, we design a parallel memory lookup scheme by incorporating a parallel CRC compression mechanism to reduce the table sizes. The multi-cycle implementation requires only an average of 1.6 memory accesses for each lookup request. This lookup latency is

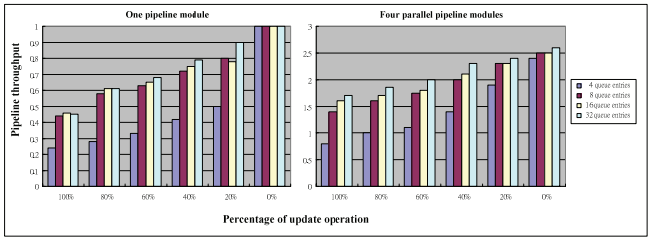


Figure 11 Throughput of the route lookup engine

achieved without increasing the memory requirement compared with the previous works.

We further present the pipeline design for the proposed lookup/update scheme. The five-stage pipeline structure naturally fits with the lookup/update process with the deployment of a queue stage which buffers lookup requests to allow table updates. Performance simulation shows that the number of queue entries significant affects the lookup throughput when frequent table updates happen. Using the TSMC 0.25u technology, the proposed PML pipeline engine with an eight-entry queue stage has achieved a clock frequency of 100 MHz which equivalently translates to a maximum of 100×10^6 lookups per second. The lookup rate can be further increased by using multiple pipelines. The evaluated four-pipeline configuration improves the throughput by a factor of 2.5 for sparse update arrivals. With multiple-pipelines, the impact of updates on the throughput is reduced because lookups and updates can perform at the same time in different pipelines. This paper has demonstrated a viable pipeline design that is scalable both to the lookups and update requests for the IPv6 system.

As so far, the proposed PML supports unicast address. However, it is easy to extend the proposed PML to multicast address. If the multicast address is detected, the Group ID address is used to perform the address lookup. For the lookup of Group ID address of the IPv6 multicast address, an additional PML set is required and the Group ID address is segmented to perform parallel address lookup.

Acknowledgments

This work was supported by the National Science Council (NSC), Taiwan, under Contract NSC94-2220-E-006-004.

References

[1] K. Etzel, "Answering IPv6 Lookup Challenges," Available at http://www.commsdesign.com/news /showArticle.jhtml?articleID=51200476.

[2] M. Wang, S. Deering, T. Hain, and L. Dunn, "Non-Random Generator of IPv6 Tables," in the *Proceedings of the IEEE Symposium on High-Performance Interconnects*, August 25-27, 2004, pp. 35-40.

[3] P. Hinden and S. Deering, "IP Version 6 Addressing Architecture," RFC 2373, IETF, Available in http:// www.ieft.org, July 1998.

[4] M.A. Ruiz-Sanchez, E.W. Biersack, and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithm," *IEEE Network*, Vol. 15, No. 2, March-April 2001, pp. 8-23.

[5] D. Pao, C. Liu, A. Wu, L. Yeung, and K.S. Chan, "Efficient Hardware Architecture for Fast IP Address Lookup," *IEE Proceedings on Computers and Digital Techniques*, Vol. 150, No. 1, January 2003, pp. 43-52.

[6] R. C. Chang and R.-H. Lim, "Efficient IP Routing Table VLSI Design for Multigigabit Routers," *IEEE Transactions on Circuits and Systems-I*, Vol. 51, No. 4, April 2004, pp. 700-708.

[7] P.A. Yilmaz, A. Belenkiy, N. Uzun, N. Gogate, and M. Toy, "A Trie-based Algorithm for IP Lookup Problem," in *IEEE Global Telecommunications Conference*, Vol. 1, 2000, pp. 593-598.

[8] R. Sangireddy, N. Futamura, S. Aluru, and A. K. Somani, "Scalable, Memory efficient, High-speed IP Lookup Algorithms," *IEEE/ACM Transactions on Networking*, Vol. 13, No. 4, August 2005, pp. 802-812.

[9] W. Doeringer, G. Karjoth, and M. Nassehi, "Routing on Longest-Matching Prefixes," *IEEE/ACM Transactions on Networking*, Vol. 4, No. 1, February 1996, pp. 86-97.

[10] B. Lampson, V. Srinivasan, and G. Varghese, "IP Lookups Using Multiway and Multicolumn Search," *IEEE/ACM Transactions on Networking*, Vol. 7, No. 3, June 1999, pp. 324-334.

[11] M. Waldvogel, G. Varghese, J. Turner, and B, Plattner, "Scalable High Speed IP Routing Lookups," in the *Proceeding of ACM SIGCOMM*, September 1997, pp. 25-36.

[12] X.o Yao, L. Li, and G. Hu, "A Fast IPv6 Route Lookup Algorithm with Hash Compression," *IEEE Communications, Circuits and Systems Conference*, Vol. 1, 2004, pp. 674-677.

[13] H. Lim, J.-H Seo, and Y.-J. Jung, "High Speed IP Address Lookup Architecture Using Hashing," *IEEE Communication Letters*, Vol. 7, No. 10, October 2003, pp. 502-504.

[14] Z. Wang, H. Wang, and Y. Sun, "High-Performance IPv4/IPv6 Dual-Stack Routing Lookup," in *IEEE International Conference on Advanced Information Networking and Applications*, 2004, pp. 476-481.

[15] T. Hayashi and T. Miyazaki, "High-Speed Table Lookup Engine for IPv6 Longest Prefix Match," in *IEEE Global Telecommunications Conference*, 1999, pp. 1576-1581.

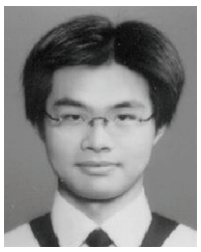
[16] N.-F. Huang, W.-E. Chen, J.-Y. Luo, and J.-M. Chen, "Design of Multi-field IPv6 Packet Classifiers Using Ternary CAMs," in *IEEE Global Telecommunications Conference*, Vol. 3, 2001, pp. 1877-1881.

[17] K. Pagiamtzis and A. Sheikholeslami, "Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey," *IEEE Journal of Solid-State Circuits*, Vol.41, No.3, March 2006, pp.712-727.

[18] P. Gupta, S. Lin, and N. Mckeown, "Routing Lookups in Hardware at Memory Access Speeds," in the *Proceeding of IEEE INFOCOM*, April 1998, pp. 240-47.

Biographies

Chung-Ho Chen received the M. S. degree in electrical engineering from the University of Missouri-Rolla in 1989, and the Ph. D. degree in electrical engineering from the University of Washington, Seattle, U.S.A. in 1993. He is currently a Professor with the Department of Electrical Engineering at National Cheng-Kung University, Tainan, Taiwan. His research areas include advanced computer architecture, SoC testing, video technology, and network storages. He co-holds a U.S. patent on a multi-computer cluster-based processing system and a R.O.C. patent on a multiple-protocol storage structure. Dr. Chen was the Technical Program Chair of the 2002 VLSI Design/CAD Symposium held in Taiwan. He is a member of the IEEE.



Chao-Hsien Hsu received the B.S. degree in electrical engineering from the National Cheng-Kung University in 2003, and the M.S. degree in computer and communication engineering from the National Cheng-Kung University in 2005. Presently, he is a R&D engineer in worldwide-headquarter of Zyxel Communication Corpora- tion. He focuses on the production line of FTTx.

Chen-Chieh Wang received a B.S. degree in electrical engineering with a minor in computer science from the Feng-Chia University, Taiwan, and an M.S. degree in computer and communication engineering from the National Cheng-Kung University, Taiwan, in 2003 and 2005, respectively. He is currently pursuing the Ph.D. degree in the Institute of Computer and Communication Engineering, National Cheng-Kung University, Taiwan. His research interests include network storages, network security, and SoC integration.