A Pipeline IP Lookup Architecture with Random Duplicate Allocation

Yi Wu and Ge Nong

***Abstract*—The gap between high throughput demand of Internet traffic and low speed capacity of a router's interface has become a bottleneck for packet forwarding. One way to close the gap is to employ a parallel mechanism, where the route lookups of multiple packets are processed simultaneously, yielding a substantial improvement in the system's throughput. This paper proposes a new pipelined trie-based routing architecture with multiple memory blocks, in which a routing table is organized as a prefix trie and the latter is further decomposed into a main trie and multiple subtries containing the lower-level and higher-level nodes, respectively. Further, the main trie is converted into an index table and the subtries are evenly distributed into all the memory blocks. A storage management technique called random duplicate allocation (RDA) is employed to balance the storage demands among all the memory blocks. Specifically, for each subtrie, the root node is stored in a randomly selected memory block, and the descendant nodes are stored in the subsequent memory blocks level by level, in a circular manner of one block for a level. The results of computer simulation experiments indicate that the routing system's aggregate throughput grows almost linearly proportional to the number of memory blocks.**

***Index Terms*—IP lookup, pipeline architecture, prefix trie, performance evaluation.**

I. Introduction

The task of looking up the route of an IP packet is to find the longest matching prefix (LMP) of the packet's destination address with the route entries in the routing table. As a common choice in practice, various prefix trie based routing architectures have been widely adopted to build high speed IP routing engines [1]–[11]. In this article, our recent attempt for designing a new trie-based pipeline IP lookup architecture with multiple memory blocks is presented, where the storage of a prefix trie is controlled by a randomization technique called *random duplicate allocation* (RDA) previously proposed for managing parallel disks in [12].

A. Related Work

To scale the aggregate throughput of a routing system, a pipeline mechanism is favorable for executing multiple searching tasks in parallel. Obviously, a high efficiency can be achieved when each pipe stage is kept busy as often as possible. Recently, a number of new progresses were reported for constructing high-performance pipelined routing systems [13]–[20]. These works can be classified into two categories according to the storage design and access strategy, respectively. Baboescu et al. [18], Kumar et al. [19] and Kim et al. [13] organize the pipeline in a linear/circular fashion, where each node of the prefix trie resides on the subsequent stage following the one that their parent node is located in. Thus, all the nodes on a same level of the prefix trie are stored in a same pipe stage, and a packet can start a route lookup from the initial stage and step through the subsequent stages one by one. However, it is observed that the lengths of the prefix entries in a routing table are distributed unevenly. Especially, the majority of a routing table consists of 24-bit prefix entries [21], [22], resulting in unbalanced storage and bandwidth demands for different memory blocks. Distinct from the existing work, we recently proposed in [20] a scalable routing architecture for prefix tries, where a randomization technique is applied to balance the workload for each block. In more details, each node of the prefix trie is stored into a randomly selected memory block. The scheduling of memory accesses in this architecture was modeled as a bipartite matching problem, and has been recognized as the most challenging part to be solved in practice.

Yi Wu and Ge Nong (corresponding author) are with the Department of Computer Science, Sun Yat-sen University, Guangzhou, China. Email: issng@mail.sysu.edu.cn.

B. Our Proposed

Following the ideas previously proposed for external memory algorithms in [12], [23], [24], we herein present a novel pipelined routing architecture, which can support a wide set of trie-based routing algorithms. Regardless of the concrete design of each individual trie-based routing algorithm, the underlying prefix trie is generally separated into two parts, saying that a main trie and a set of subtries, respectively. The main trie contains the first several levels of the prefix trie and is mapped to an index table, while a subtrie consists of a predefined set of higher-level nodes and is stored in the memory blocks level by level, in a circular manner of one level per block. Specifically, the root node of each subtrie is randomly distributed into a memory block with a uniform probability, while the descendant nodes are cyclically stored in the subsequent memory blocks in a fashion of one level per block. Besides, the technique of node redundancy is also applied in a more restricted way, in terms of that only two copies of the prefix trie are stored in the memory blocks to reduce memory access conflicts.

C. Differences

The idea of pipeline circulation and trie partition has been widely employed in designing high-performance routing engines. Baboescu et al. proposed a multi-point pipeline engine in [18], where the pipe stages are organized in a circular fashion. For purpose of storage balance, the prefix trie is split

978-1-4673-1544-9/12/$31.00 ©2012 IEEE

into multiple subtries of similar sizes, and a heuristic strategy is used to evenly distribute these subtries into all the memory blocks. The approach in [18] was further extended by Kumar et al. in [19], to propose another circular pipeline architecture called Circular Adaptive and Monotonic Pipeline (CAMP), in which the prefix trie is partitioned into a root subtrie and multiple leaf subtries of similar sizes. To be specific, the root subtrie constitutes a hash table of 2^r entries, each for a pipe stage, where r denotes the highest r bits of an IP address. Hence, for each subtrie, its root node is assigned to a pipe stage selected by the highest r bits, while the child nodes are mapped to the subsequent stages following the root stage in a circular manner of one level per stage.

Our architecture proposed here also uses the idea of pipeline circulation and trie partition, however, it *differs* from the prior arts as follows. We exploit the use of a technique called random duplicate allocation (RDA), which was formerly proposed for storage management in parallel disks [12], to improve the system's performance by scheduling memory accesses with less conflicts. In more details, we maintain only two copies for each subtrie, where the storage of each copy is allocated independently. Hence, to access the routing information stored in a subtrie, we can traverse any of the two copies of the subtrie, resulting in an increased success probability. The philosophy for us to employ RDA is that for building a high-performance routing system, increasing the capacity of each individual memory is far more easy, cheap and feasible than increasing the number of memory blocks.

The rest of this paper is organized as follows. Section II presents the proposed routing system. Next, a brief mathematical analysis is sketched in Section III to analyze the system's efficiency. Further, a set of simulation experiments are conducted to investigate the system's performance in Section IV. Finally, Section V gives the conclusion.

II. Routing System

For presentation convenience, we have the following notations used in the rest of this article. When we say a packet's subtrie, it refers to the subtrie which is traversed to retrieve the routing information of this packet. For each subtrie, the memory block where the subtrie's root resides is called the subtrie's root block. Further, for a packet, the root block of its subtrie is also referred as the packet's root block.

A. Architecture

Fig. 1 depicts the proposed routing architecture, which consists of a fast memory (e.g. SRAM) for storing the index table converted from the main trie, multiple high-capacity memory blocks (e.g. DRAMs) for storing the subtries, a routing buffer for queuing packets, a scheduler for controlling the routing processes of queuing packets, and a packet filter to control that no any two queuing packets in the routing buffers

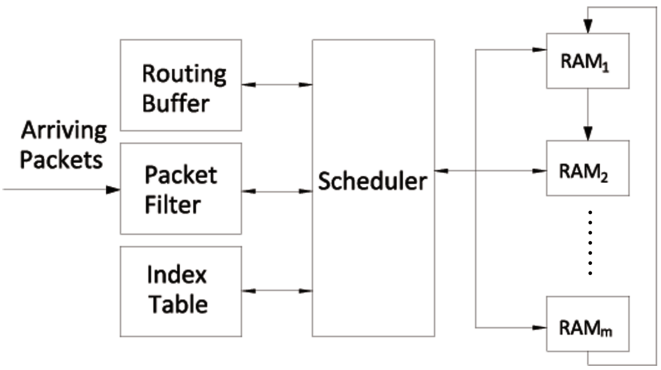


Figure 1. The routing architecture.

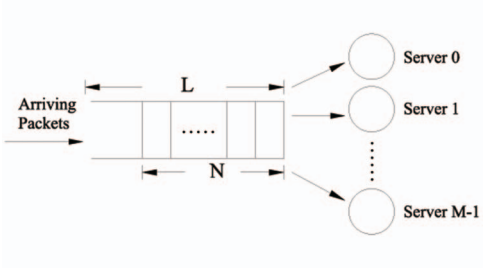
are of a same destination address.[1]

The main trie is represented as the index table stored in a fast but small memory. Starting at the root block, the nodes of each subtrie are stored in the successive memory blocks cyclically, in a manner of one level per block. For example, if the root node is stored at memory block i, then the nodes of level j will be stored in memory block $(i+j) \mod M$.

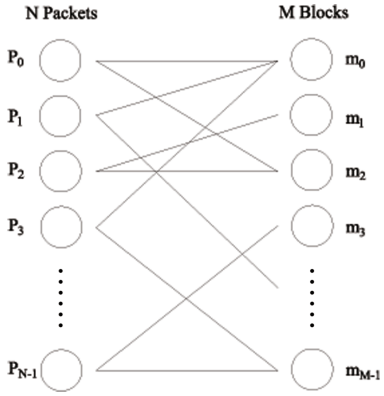
For each newly arriving packet, two routing stages will be experienced to find the LMP for the packet's destination addresses, i.e. a single-step matching stage in the index table followed by a searching stage that may span over multiple memory blocks. The index table consists of 2^r entries, where each entry is composed of a prefix field and a node pointer. Using the index table, the route lookup of a packet is performed as follows. First, the highest r bits of the packet's destination address is extracted to index an entry i in the index table. If the node pointer of entry i is null, then the route lookup is done; or else a subtrie with its root location indicated by the node pointer will be further traversed to complete the route lookup procedure.

To achieve a high aggregate throughput of the memory blocks, the scheduling objective in this routing architecture is *to select a maximal set of queuing packets with different root blocks*. Recalling that for each subtrie, its nodes are cyclically stored in the successive memory blocks following its root block, in a manner of one level per block. Therefore, traversing a subtrie for route lookup of a packet will start by accessing the root, then further to access the other nodes on deeper levels of the subtrie, one node per level and one level per block, until a leaf node is reached. Hence, the selected subtries with conflict-free root blocks can be traversed in parallel as *a batch*, resulting in an increased efficiency of the multi-block memory system. The term "a batch" means that the traversals of the selected subtries start at a same time slot, and terminate until all the traversals have finished, regardless of that they may finish in different time slots.

[1]The packet filter is employed for two purposes: (1) to avoid routing redundancy, for any two packets of a same destination address will be routed to an identical output port; (2) to maintain the FIFO order of packets with a same destination.



(a) Queuing model.



(b) Bipartite matching model.

Figure 2. The queuing and bipartite matching models.

B. Queuing Model

For the described routing architecture, we show in Fig. 2(a) a queuing model for mathematical analysis. In this model, a constant-time query to the index table will be performed to lookup the root block of each newly arriving packet, then the packet is enqueued at the routing buffer. The symbols used in this model are defined below:

- M: the number of memory blocks. The memory blocks are denoted as $m_0, m_1 ..., m_{M-1}$. For presentation simplicity, M is assumed to be a power of 2.
- L: the routing buffer's capacity in unit of packets, i.e. at most L packets queuing in the buffer.
- N: the number of queuing packets in the routing buffer.

In this queuing model, we also hold the following assumptions for analysis:

- Time in the system is split into fixed-size slots, where a time slot is defined as a memory's access cycle. Furthermore, each time slot is equally divided into M cycles.
- Each arriving packet is of a fixed length. At most one packet per cycle can arrive at the routing buffer, i.e. at most M arriving packets per time slot.
- Each memory block can be accessed at most once per time slot.
- Each subtrie has two identical copies rooted at two randomly chosen memory blocks, respectively.

C. Scheduling

For each newly arriving packet, the index table is queried by the first r bits of the packet's destination address to locate the root of the packet's subtrie, then the subtrie is traversed level by level to route the packet. To traverse a subtrie, a set of memory blocks are requested for accesses. In this sense, a search request is generated for each queuing packet. To denote a search request with accesses to memory blocks m_i, m_j, m_k for example, we use a tuple $R(s_i, s_j, s_k)$. That is, if memory block i needs to be accessed, s_i will be in the $R(...)$ tuple.

A routing process for packets in the routing buffer is split into two phases: scheduling and accessing. The former is to select a set of packets with different root blocks, and the latter is to traverse the subtries for routing the selected packets. As the scheduling output, we have a set of packets whose searching requests can be executed in parallel. Then in the accessing phase, each selected packet will access the corresponding memory blocks for traversing its subtrie. In more details, without loss of generality, suppose R_i, R_j are the searching requests from any two selected packets, and M_i, M_j are the initial memory blocks to be accessed by these two requests. During the accessing phase, R_i will cyclically visit the memory blocks numbered $m_i, m_{(i+1) \bmod M}, m_{(i+2) \bmod M}$..., until a leaf is reached, so does R_j. Consequently, the executions of R_i and R_j are conflict-free in the whole accessing phase, and hence can be parallelized.

The arbitration problem in the scheduling phase can be cast as bipartite matching in Fig. 2(b), where the two parties are the N queuing packets and the M memory blocks, and an edge between a packet and a memory block indicates that a copy of the packet's subtrie is rooted at the latter. Notice that a subtrie has two copies rooted at two different memory blocks, in order to traverse this subtrie for routing a packet, it is sufficient to traverse any of these two copies. Hence, each packet vertex in the graph has two edges. Generally speaking, any bipartite matching algorithm can be employed in the scheduling phase to select a set of packets satisfying the given constraints, e.g. PIM in [25], iSLIP in [26] and etc.

To help understanding the routing process, we give in Table I an example for scheduling packets and accessing memory blocks, where the arbitration policy at each memory block is FCFS (first come first served), $M = 4$ and initially $N = 0$. As we assumed in Section II-B, each time slot is divided into M cycles, and at most one packet per cycle can arrive at the routing buffer.

- In time slot 0, $P_0 \sim P_3$ arrive at the routing buffer, and memory access requests $R_0 \sim R_3$ are generated for them, respectively, where R_0 for P_0, R_1 for P_1 and so on. Because both R_2 and R_3 request m_2, an access conflict will happen and must be resolved by the scheduling algorithm. Since the two packets are from a same routing queue, governed by FCFS, R_2 will be granted to P_2 which is elder and hence of a higher priority, resulting in that the packets $\{P_0, P_1, P_2\}$ of access requests $\{R_0, R_1, R_2\}$ are selected. The requests

Table I
A SCHEDULING EXAMPLE

(a) Arrivals

Initial State ($M = 4, N = 0$)			
Packet	Time Slot	Cycle	Searching Request
P_0	0	0	$R_0(m_0, m_1, m_2)$
P_1	0	1	$R_1(m_1, m_2)$
P_2	0	2	$R_2(m_2, m_3)$
P_3	0	3	$R_3(m_2, m_3)$
P_4	1	4	$finished$
P_5	1	5	$R_5(m_3, m_0, m_1)$
P_6	1	6	$R_6(m_0, m_1, m_2)$
P_7	1	7	$R_7(m_2, m_3)$
P_8	2	8	$R_8(m_1)$
P_9	2	9	$R_9(m_3, m_0, m_1)$
P_{10}	2	10	$R_{10}(m_0, m_1, m_2)$
P_{11}	2	11	$R_{11}(m_3, m_0, m_1)$

(b) Scheduling

Time Slot	Action	Selected Request	Queuing Request
0	schedule	$\{R_0, R_1, R_2\}$	$\{R_3\}$
1	access	$\{R_0, R_1, R_2\}$	$\{R_3, R_5, R_6, R_7\}$
2	access	$\{R_0, R_1, R_2\}$	$\{R_3, R_5, R_6, R_7, R_8, R_9, R_{10}, R_{11}\}$
3	access	$\{R_0\}$	$\{R_3, R_5, R_6, R_7, R_8, R_9, R_{10}, R_{11}\}$
4	schedule	$\{R_3, R_5, R_6, R_8\}$	$\{R_7, R_9, R_{10}, R_{11}\}$
5	access	$\{R_3, R_5, R_6, R_8\}$	$\{R_7, R_9, R_{10}, R_{11}\}$
6	access	$\{R_3, R_5, R_6\}$	$\{R_7, R_9, R_{10}, R_{11}\}$
7	access	$\{R_5, R_6\}$	$\{R_7, R_9, R_{10}, R_{11}\}$
8	schedule	$\{R_7, R_9, R_{10}\}$	$\{R_{11}\}$
9	access	$\{R_7, R_9, R_{10}\}$	$\{R_{11}\}$
10	access	$\{R_7, R_9, R_{10}\}$	$\{R_{11}\}$
11	access	$\{R_9, R_{10}\}$	$\{R_{11}\}$
12	schedule	$\{R_{11}\}$	$\{\}$
13	access	$\{R_{11}\}$	$\{\}$
14	access	$\{R_{11}\}$	$\{\}$
15	access	$\{R_{11}\}$	$\{\}$

$\{R_0, R_1, R_2\}$ are conflict-free and performed in parallel as *a batch* in the accessing phase consists of time slots 1-3. The length of the accessing phase is determined by R_0 that access the most memory blocks.

- At the beginning of time slot 4, another scheduling phase begins and $\{R_3, R_5, R_6, R_8\}$ are selected. These requests will occupy the next 3 time slots, i.e. times 5-7, to execute memory accesses. Similarly, $\{R_7, R_9, R_{10}\}$ and $\{R_{11}\}$ are chosen by the scheduling instances at time slots 8 and 12, respectively, and both occupy 3 time slots, i.e. time slots 9-11 and 13-15, respectively. Notice that there's no need to produce R_4, because the route of P_4 is determined immediately by the quick match in the index table.

Notice from the example, the routing process of a packet consists of two phases: a scheduling phase followed by an accessing phase, where the former occupies one time slot and the latter lasts for one or more time slots. During the scheduling phase, a set of queuing packet are chosen for route lookups. Then, the searching request of each chosen packet takes a number of consecutive time slots to complete its lookup. From this example, we can see that in order to further improve the system's efficiency, any two consecutive routing processes can be pipelined by aligning the scheduling phase of one with the accessing phase of another. However, for presentation simplicity, this example assumes that the routing processes are performed in sequence without overlapping.

III. MATHEMATICAL ANALYSIS

The independent disk model (IDM) proposed in [23] is used to analyze parallel I/O operations on disks, which consists of a processor, multiple disks and a RAM. During each I/O step, one data block can be read from each disk synchronically and multiple conflict-free reads can be performed in parallel. For efficiently emulating an IDM with multi-head disks, a technique called random duplicate allocation (RDA) was proposed in [12]. Several parameters are used in the analysis:

- D: the number of external disks. The disks are denoted as $d_1, d_2, ..., d_D$, respectively.
- N: the number of data blocks to be read from the disks. Each data block has two copies in any two disks, once copy per disk. More exactly, the two copies of a data block n_i are randomly allocated in the disks numbered u_i, v_i, where $u_i, v_i \in \{d_1, d_2, ..., d_D\}$ and $u_i \neq v_i$.
- H: the number of parallel reads for fetching the batch of N data blocks, or in other words, the maximal number of accesses to one disk during the process of reading N data blocks from D disks. Suppose h_i is the number of data blocks read from d_i, then we have $\sum_{i=1}^{D} h_i = N$ and $H = \max\{h_1, h_2, ..., h_D\}$.

The key idea of the analysis for RDA in [12] is summarized as follows. The problem is: there are a batch of N blocks to be accessed, each block has two copies and each copy is randomly stored in a disk, we would like to determine the minimal number of parallel read operations for reading these N blocks. The problem in concern can be modeled as an undirected allocation multigraph G_a, where $G_a = \{(d_1, d_2, ..., d_D), (\{u_1, v_1\}, ..., \{u_N, v_N\})\}$. Specifically, each vertex represents a disk and an undirected edge $\{u_i, v_i\}$ indicates that both u_i and v_i have a copy for n_i. A valid schedule for the batch is a directed subgraph G_s of G_a, where a directed edge $\{u_i, v_i\}$ indicates that block n_i is read from u_i. The scheduling objective is to find an optimal subgraph $G_{s\_optimal}$, H of which is no more than that of any other valid schedule. It is proven in [12] that given a batch of N randomly and duplicately allocated data blocks to be read from D disks, no more than $\lceil N/D \rceil + 1$ steps will be needed *with high probability*. We will show that this result can be applied to our proposed routing architecture too.

Recalling that each subtrie has two copies and each copy's root node is randomly stored in a memory block, a search request can start from one of the two root memory blocks and step cyclically through the subsequent memory blocks for executing the route lookup. The scheduler selects a maximal matching between the memory blocks and the queuing packets during each scheduling phase. Hence, given M memory blocks and N queuing packets, the scheduling problem can also be cast as an undirected multigraph $G_a = \{(m_1, m_2, ..., m_M), (\{u_1, v_1\}, ..., \{u_N, v_N\})\}$, where

u_i, v_i are the root memory blocks for packet i and $u_i, v_i \in \{m_1, m_2, ..., m_M\}$. Similarly, the scheduling objective is to find a directed subgraph G_s, where the number of parallel memory accesses H is no more than that of any other valid schedule. In this way, we can come to the conclusion that, with high probability, no more than $(\lceil N/M \rceil + 1) \times S_{max}$ time slots are needed to fully execute these N searching requests, where S_{max} is the greatest search depth of these N search requests.

IV. Simulation Experiments

A series of simulation experiments are conducted to evaluate the performance of our proposed routing system for pipelining the prefix trie algorithms of binary trie (BT) and fixed-stride trie (FST) [5]. Each simulation result is the mean of the samples from 10 runs, where each run has 100,000 packet arrivals. For each newly arriving packet at the routing buffer, we randomly choose one from the 397139 entries of the AS6447 BGP routing table [22] as the packet's destination address. Further, the packet arrivals are governed by a Bernoulli process.

A. Parameter Description

The parameters in use are described below:

- S: the average number of steps for looking up the route of a packet, each step needs to access a memory block once. In more details, each packet has a searching request, and each searching request consists of one or multiple steps, one memory access per step. Each memory can perform at most one access per time slot.
- R: the number of copies for each subtrie stored in the memory blocks. Only two values for R are investigated here, i.e. 1 or 2.
- X: the allocation mode of a subtrie's copies. This parameter is valid only for $R \geq 2$. If $X = 0$, each copy of a subtrie is randomly rooted at a memory block (in this case, two copies of a subtrie may be rooted at a same memory block); or else for $X = 1$, any two copies of a subtrie must be rooted at two different memory blocks.
- Q: the average number of queuing packets in the routing buffer, at the beginning of a time slot.
- W: the normalized workload on a memory block, which is defined as $W = \lambda$, where λ is the mean traffic arrival rate at the routing buffer in each cycle. In more details, a packet arrives with a probability of λ at the beginning of each cycle. Notice that a time slot is composed of M cycles, the mean traffic arrival rate at each time slot is therefore $M\lambda$.

B. Scheduling Algorithm

The bipartite matching model cast in Section II-C is employed in the scheduling phase to select packets. Specifically, at the beginning of each scheduling instance, all the queuing packets and memory blocks are initially unmatched, and the following two steps are repeated until a maximal matching is established:

1) Each unmatched queuing packet randomly chooses one from its unmatched root memory block(s) to request;
2) Each unmatched memory block chooses the eldest from all its requesting packets to match.

C. Simulation Analysis

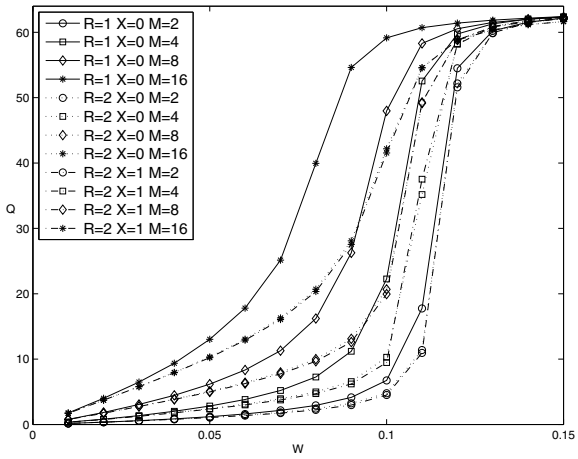
The investigated performance measure is the average buffer length Q as a function of the normalized workload W. Given the buffer size $L = 64$, Fig. 3(a)-3(b) depict the system performance with varying R, X and M. Obviously, each curve gradually increases at the very beginning when W increases, and grows rapidly when the system is going to be overloaded, where W approaches a point termed as the threshold of W. As can be seen, the curves for $R = 1$ are always above their counterpart for $R = 2$, which means that a much higher efficiency can be achieved by using only one more copy for each subtrie. However, the change on X has only negligible impacts on the system's performance.

In Fig. 3, the threshold of W gets smaller as M increases for the pipelined BT and 6-FST. The reason is that the average utilization of each memory block sharply decreases when M becomes larger and the buffer size L keeps unchanged. In contrast, Fig. 4 indicates that an increase in the size of the routing buffer can alleviate the performance degradation caused by a large M. This can be explained as following. Suppose that there are many queuing packets in the routing buffer, it is more likely for a memory block to get matched with a packet when M is smaller. However, when M becomes larger, given the same number of queuing packets, the probability for a memory block to get matched with a packet will be decreased, leading to a decline in the system's performance. Hence, if L is increased, more queuing packets will take part in the scheduling and the probability for a memory block to be matched will be increased, resulting in an increased utilization for each memory.

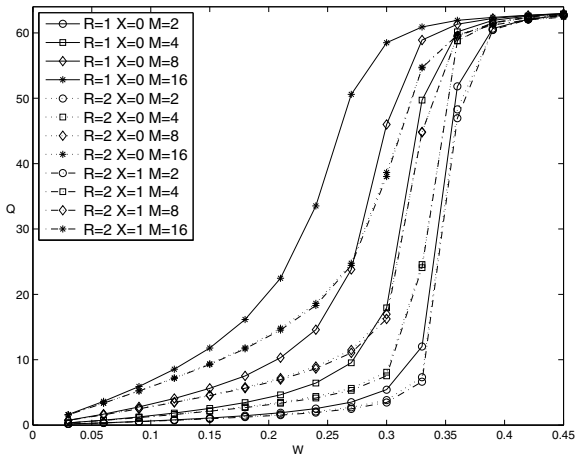
Provided $R = 2$ and $X = 1$, Fig. 4(a)-Fig. 4(b) show the performance of the pipelined BT and 6-FST with varying M and L, where L is a multiple of M. We can see that a performance degradation occurs in between $L = 4M$ and $L = 8M$, indicating that $L = 32$ is large enough for $M = 4$ to achieve a nearly optimal performance. In the experiments, the statistics values of S for 6-FST and BT are 2.47 and 7.6, respectively. From these figures, we observed that the threshold of W for the pipelined 6-FST is about 3 times of that for the pipelined BT, which means that the larger S and the smaller threshold of W.

V. Summary

We have proposed in this paper a pipeline routing architecture that can be applied to a broad range of trie-based routing algorithms, e.g. BT, FST and etc. Specifically, the technique of random duplicate allocation previously proposed for controlling parallel disks [12] is exploited to improve the pipeline efficiency of this routing architecture. The results of simulation experiments revealed that using only one more copy for the prefix trie, the routing system's throughput can

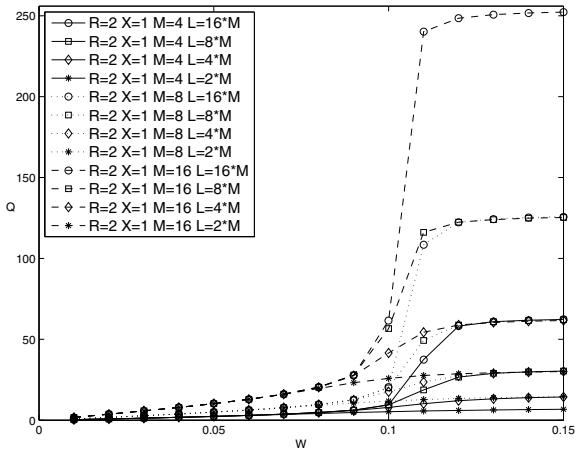


(a) Pipelined BT with varying R, M and X

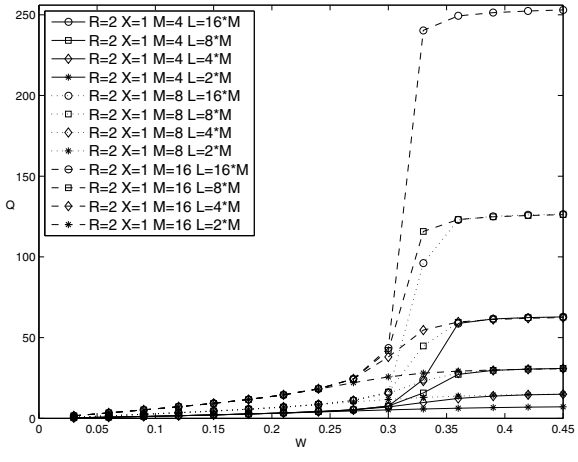


(b) Pipelined 6-FST with varying R, M and X

Figure 3. The average buffer length as a function of the normalized workload, i.e. Q vs. W, for the pipelined BT and 6-FST algorithms, with $L = 64$, $R \in \{1, 2\}$, $M \in \{2, 4, 8, 16\}$ and $X \in \{0, 1\}$.



(a) Pipelined BT with varying M and L



(b) Pipelined 6-FST with varying M and L

Figure 4. The average buffer length as a function of the normalized workload, i.e. Q vs. W, for the pipelined BT and 6-FST algorithms, with $R = 2$, $X = 1$, $M \in \{4, 8, 16\}$ and $L \in \{2M, 4M, 8M, 16M\}$.

be improved significantly, due to a substantial decrease in memory access conflicts.

To generalize the key idea of this work, we are currently extending the scheme to support other trie-based routing algorithms where each entry of the routing table doesn't has a one-to-one mapping with a node in the prefix trie, e.g. Prefix Trie (PT) [1] and Multi-Prefix Trie (MPT) [4]. Besides, it is also noticed that using two copies for a prefix, the update cost will directly be doubled. Fortunately, if we always rooted the two copies of a subtrie at two different memory blocks, i.e. $X = 1$, the updating operations of both copies can be parallelized for speed acceleration. Further, this pipeline system may suffer from a low utilization caused by uneven heights of the subtries to be traversed for routing the selected packets. One possible way to solve this problem is to take into account the height of each subtrie when scheduling the queuing packets. These issues are worthy of being investigated in a further study.

VI. Acknowledgment

This research work was supported by the National Natural Science Foundation of China through grant 60873056 and the Fundamental Research Funds for the Central Universities of China through grants 11lgzd04 and 11lgpy93.

References

[1] M. Berger, "IP lookup with low memory requirement and fast update," in *Proc. IEEE HPSR*, Jun 2003, pp. 287–291.

[2] V. Srinivasan and G. Varghese, "Faster IP lookups using controlled prefix expansion," *ACM Trans. on Computer Systems*, pp. 1–40, Feb 1999.

[3] S. Nilsson and G. Karlsson, "IP-address lookup using lc-tries," *IEEE J. on Selected Areas in Communications*, pp. 1083–1092, Jun 1999.

[4] S. Hsieh, Y. Huang, and Y. Yang, "A novel dynamic router-tables design for IP lookup and update," in *IEEE Int. Con. on Future Information Technology*, May 2010, pp. 1–6.

[5] S. Sahni and K. Kim, "Efficient construction of multibit tries for IP lookup," *IEEE/ACM Trans. on Networking*, pp. 650–662, Aug 2003.

[6] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small forwarding tables for fast routing lookups," in *Proc. ACM SIGCOMM*, 1997, pp. 3–14.

[7] S. Sahni and K. Kim, "Efficient construction of fixed-stride multibit tries for IP lookup," in *IEEE Workshop on Future Trends of Distributed Computing Systems*, Nov 2001, pp. 178–184.
[8] ——, "Efficient construction of variable-stride multibit tries for IP lookup," in *Proc. Symp. on Applications and the Internet*, Feb 2002, pp. 220–227.
[9] Y. Chang, Y. Lin, and C. Su, "Dynamic multiway segment tree for IP lookups and the fast pipelined search engine," *IEEE Trans. on Computers*, pp. 492–506, Feb 2010.
[10] H. Lim, C. Yim, and E. Swartzlander, "Priority tries for IP address lookup," *IEEE Trans. on Computers*, pp. 784–794, June 2010.
[11] W. Lu and S. Sahni, "Recursively partitioned static IP router-tables," *IEEE Trans. on Computers*, pp. 1683–1690, Dec 2010.
[12] P. Sanders, S. Egner, and J. Korst, "Fast concurrent access to parallel disks," in *Proc. 11th annual ACM-SIAM symposium on Discrete algorithms*, 2000, pp. 849–858.
[13] K. Kim and S. Sahni, "Efficient construction of pipelined multibit-trie router-tables," *IEEE Trans. on Computers*, pp. 32–43, Jan 2007.
[14] A. Basu and G. Narlikar, "Fast incremental updates for pipelined forwarding engines," *IEEE/ACM Trans. on Networking*, pp. 690–703, Jun 2005.
[15] M. Bando and H. J. Chao, "Flashtrie: Hash-based prefix-compressed trie for IP route lookup beyond 100Gbps," in *Proc. IEEE INFOCOM*, March 2010, pp. 1–9.
[16] W. Jiang and V. Prasanna, "A memory-balanced linear pipeline architecture for trie-based IP lookup," in *IEEE Symp. on High-Performance Interconnects*, Aug 2007, pp. 83–90.
[17] W. Lu and S. Sahni, "Packet forwarding using pipelined multibit tries," in *Proc. IEEE Symp. on Computers and Communications*, June 2006, pp. 26–29.
[18] F. Baboescu, D. Tullsen, G. Rosu, and S. Singh, "A tree based router search engine architecture with single port memories," in *Proc. IEEE Int. Symp. on Computer Architecture*, June 2005, pp. 4–8.
[19] S. Kumar, M. Becchi, P. Corwley, and J. Turner, "CAMP: fast and efficient IP lookup architecture," in *ACM/IEEE Symp. on Architecture for Networking and Communications Systems*, 2006, pp. 51–60.
[20] Y. Wu and G. Nong, "A scalable routing architecture for prefix tries," in *Proc. 17th IEEE Int. Con. on Networks*, Dec. 2011.
[21] G. Huston, *Analyzing the Internet's BGP routing table*, July 2003.
[22] "BGP routing table analysis reports," 2007.
[23] J.Vitter and E.Shriver, "Algorithms for parallel memory I: Two level memories." pp. 110–147, 1994.
[24] R. Barve, E. Grove, and J. Vitter, "Simple randomized mergesort on parallel disks," *Paral. Comput.*, pp. 601–631, 1997.
[25] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "high-speed switch scheduling for local area networks," *ACM Trans. Comput. Syst.*, pp. 319–352, Nov 1993.
[26] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. on Networking*, pp. 188–201, April 1999.