A Scalable Routing Architecture for Prefix Tries

Yi Wu and Ge Nong*

Department of Computer Science
Sun Yat-sen University, Guangzhou, PRC
Email: wu.yi.christian@gmail.com, *issng@mail.sysu.edu.cn

***Abstract*—While the throughput demand for a backbone router keeps increasing constantly, both routing and switching of packets are facing tough challenges for running faster. Currently, the prefix tries based routing algorithms are playing a key role in building high performance routing systems. We proposed in this article a routing architecture for scaling the throughput of a trie-based routing system, which consists of multiple memory blocks for trie storage and a buffer for queuing packets to resolve temporary memory access contentions. Specifically, when a trie is constructed, we store the nodes of the trie evenly into each memory block. The scheduling of queuing packets to access the memory blocks for their routes is modeled as a bipartite matching problem. A queuing model is developed to examine the system's theoretical performance under some simplified assumptions, and a series of computer simulation experiments are conducted for performance evaluation of the proposed system under more realistic conditions. The results from both the analytical queuing model and the simulation experiments indicate that this architecture can be a potential candidate for building high bandwidth routing engines for backbone routers.**

I. Introduction

A. Problem to Solve

An IP router executes two fundamental functions on packets: routing and switching. Accordingly, a packet going through the router will experience two stages in sequence, namely, the routing stage followed by the switching stage. Upon the arrival of a packet at an input of the router, the IP destination address is used to look up the route table to decide via which output the packet will be forwarded to the next hop. Given the output port has been determined, the packet can be scheduled for switching from the input to the destined output. While the throughput demand for a backbone router keeps increasing constantly, both routing and switching of packets are facing tough challenges for running faster. Of particular interest to us in this article is the design and analysis of a scalable routing architecture for backbone routers.

At each input port of an router, packets will arrive sequentially, i.e., no more than one packet will arrive at the same time. However, multiple packets with a same destination output port may arrive at different input ports simultaneously, resulting in a contention for the output port. To resolve contentions, a switching buffer is necessary to be provided in the switching stage inside a router. In a backbone router, each input port is equipped a routing engine to route its incoming packets. A common design choice for the existing routing engines is that it is able to route packets coming at the maximum arrival rate. Hence, no buffer is needed to queue packets in the routing stage. In order to scale the aggregate throughput of a routing engine, multiple memory blocks are usually employed to store the routing table for parallel memory accesses [1]–[7]. With careful arrangements, multiple conflict-free memory accesses could be performed simultaneously, resulting in an increased routing throughput. However, such a storage design is tightly coupled with the routing algorithm in use, or in other words, it is routing algorithm specific. Different from these prior arts, we propose in this article a novel scalable routing architecture using multiple memory blocks for parallel accesses and a buffer for queuing packets to resolve memory access contentions. This architecture is able to support the running of most (if not all) existing trie-based routing algorithms, e.g. Binary Trie (BT), Prefix-Tree (PT) [8], Fixed-Stride Trie (FST) [9], Multi-Prefix Trie (MPT) [10] and etc.

*To whom correspondence should be addressed.

B. Trie-Based Routing Algorithms

In an IP router, the route of a packet is searched by looking up its destination address from the entries of the route table in a manner of longest matching prefix (LMP). For this purpose, a plethora of trie-based data structures have been proposed to organize the storage of the prefixes in the route table [8]–[18]. In a trie-based routing scheme, each node of the prefix trie is linked with others via pointers. For various trie-based IP address routing algorithms, the underlying LMP searching process is analogous in principle as following. When a packet comes, the system creates a search request for its destination address. Then a searching along the trie starts from the root node and finishes when it reaches a leaf node or meets some specific conditions. As the outcome, the destined output port is known and the packet can proceed to the switching stage.

Currently, the category of trie-based routing algorithms are playing a key role in building high performance routing systems, interested readers may refer to [19], [20] for more information. To scale the throughput of a routing system, great efforts have been devoted to the study of parallel algorithms for executing multiple searching tasks simultaneously by organizing these memory blocks into a pipeline [1]–[4], [6]. For examples, Basu and Narlikar [2], Sahni and Kim [1] proposed the parallel versions of the FST originally proposed by Srinivasan and Varghese [11]. For a k-level FST, the number of distinct prefix lengths is restricted to k via controlled prefix expansion and therefore the prefix trie is divided into k levels. As a result, k memory blocks can be employed to store the

trie in a manner of one block specific for all the nodes on a level, and accessing the nodes on different levels can be pipelined. Hence, searching a route from level to level of the trie can access these memory blocks one by one in sequence, and multiple searching processes can be pipelined to improve the system's throughput.

For a pipeline routing scheme with multiple memory blocks, a high pipeline efficiency can be achieved only when most memory accesses at different pipeline stages can be parallelized. Unfortunately, the prefix lengths of IP addresses have been observed to be distributed unevenly [21], [22], this will cause that the accesses to different memory blocks are not balanced, and which in turn degrades the pipeline efficiency as well as the whole routing system's throughput. For instance, the study in [2] showed that, the 24-bit prefix entries occupy more than half of the routing table and belong to the same level of the trie, and the memory space and bandwidth required by this level is much higher than the others. Furthermore, most of the 24-bit prefix nodes are leaf nodes. This results in that most searching processes will finish in some common pipeline stage and therefore the utilizations of the subsequent stages is low. Consequently, that common stage is in fact the final stage for most searching processes and constitutes a performance bottleneck of the FST algorithm.

To overcome the problem of unbalanced memory accesses, some novel pipeline architectures have been proposed. Kim et al. [1] propose a linear-pipeline scheme employing dynamic programming and variable-stride to minimize the maximum per-stage memory, where the nodes of a prefix trie are mapped to the pipeline stages level by level. Jiang et al. [4] give an alternative that divides the original trie into a set of subtries via prefix expansion, and then distribute the nodes on each level of a subtrie into different pipeline stages. Both Baboescu et al. [6] and Kumar et al. [7] organize the pipeline in a circular fashion and split the original trie into subtries using an iterative algorithm and prefix expansion, respectively; and the root of a subtrie can be stored in any stage while the descended nodes of the subtrie are mapped into the subsequent stages level by level. In more details, the initial bits in the packet header are used as an index to the subtries stored in different pipeline stages. Thus, a search can start from any stage and wrap-around through the pipeline. Different from the existing works, we will apply a randomization technique to evenly distribute the nodes of a prefix trie into all the memory blocks without splitting the original trie in advance. Hence, a search will visit the memory blocks *randomly* rather than going through them one by one as pre-defined pipeline stages.

In the rest of this article, we will propose a scalable routing architecture and study its performance as follows. First the routing architecture is proposed together with its queuing model and a scheduling algorithm in Section II. Next, an analysis of the queuing model is developed in Section III. Further in Section IV, a series of simulation experiments are conducted to investigate the system's performance under realistic traffic samples. Finally we give the conclusion.

II. Architecture, Model and Scheduling

In this part, we describe the buffered routing architecture and its queuing model, as well as a sample algorithm for scheduling the memory accesses of queuing packets in the routing buffer.

A. Routing Architecture

Fig. 1 shows the proposed routing architecture, which consists of multiple memory blocks (typically SRAMs) for storing the trie-based routing table and a buffer for queuing packets to be routed, as well as a scheduler for controlling how the routing task for each queuing packet is executed. A high-bandwidth switching fabric, e.g. crossbar, is provided in between the buffer and the memory blocks for carrying data read from or written into the memory blocks. The nodes of the prefix trie are *evenly* distributed into all the memory blocks. That is, when a new node is added to the trie, we randomly select one from all the memory blocks to store the node (i.e., each memory block is chosen with a uniform probability). Packets arrive at the routing buffer sequentially, that is, one by one and at most one packet per time slot. The buffer supports random accesses of queuing packets and at each scheduling instance, the scheduler will select from the buffer a set of conflict-free queuing packets to look up their routing information stored in the memory blocks, one memory block per selected packet. For each individual packet, the whole process for looking up its routing information may comprise multiple memory accesses. When the memory accesses from different packets are free of conflict, they can be performed in parallel. Hence, the key for this routing architecture to achieve a high throughput is to schedule the queuing packets with conflict-free memory accesses as many as possible.

Modern backbone IP routers are commonly built on top of switches, which implies that there must be a buffer for the switching stage. The philosophy supporting our proposed routing architecture is that, if the routing buffer has a better performance (in terms of queuing delay and packet loss probability) than the switching buffer, the router will still achieve a whole system's performance comparable to that of its counterpart without the routing buffer. Notice that for any two IP packets with a common destined IP address queuing in the routing buffer, without loss of generality, they will be routed to the same destined output port. Hence, we assume that in each scheduling instance, no any two queuing packets in the scheduling set are of an identical destined IP address [1].

B. Queuing Model

In order to establish an analytical performance model for the proposed routing architecture, we make the following general assumptions.

- Time in the system is split into fixed-size slots.
- Each arriving packet at the routing buffer is of a fixed-length. At most one packet per time slot can arrive at the

[1]This assumption can be satisfied by applying a filter control on packets arriving at the routing buffer, a detail discussion on this is out of the scope of this article.

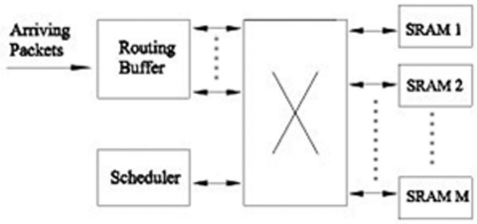


Fig. 1. The proposed routing architecture.

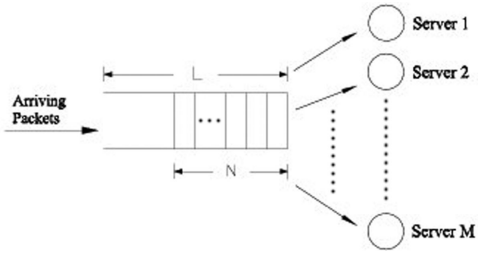


Fig. 2. The queuing model for the proposed routing architecture.

routing buffer. Furthermore, a packet will arrive only at the beginning of a time slot.

- At the end of a time slot, the scheduler will select a set of queuing packets and look up their routes from the route table distributively stored in all the memory blocks. If a selected packet completes its route lookup, it will depart the routing buffer immediately.
- Each memory block can be accessed at most once per time slot.

For denotation convenience, we introduce the following two parameters:

- N: the number of queuing packets in the routing buffer which capacity is L packets.
- M: the number of memory blocks. The memory blocks are denoted as $m_1, m_2..., m_M$. For presentation simplicity, M is assumed to be a power of 2.

Given these assumptions and notations, we show in Fig. 2 the queuing model for the proposed routing architecture. In this model, each arriving packet will generate a routing request that is composed of one or multiple memory accesses, where each memory access is counted for visiting a node of the trie on the traversing path for route lookup of the packet. To schedule the memory accesses for routing the queuing packets in the routing buffer, we employ the scheduling algorithm described in the following.

C. Scheduling Algorithm

The scheduling problem in concern can be cast as a bipartite matching graph problem. Let's define the queuing packets and the memory blocks as the graph's vertices, and an arc connecting a packet vertex and a memory vertex means that the packet has an access request to the memory. The scheduling objective is to find a maximum number of arcs satisfying that no more than one arc can connect to a node (no matter what kind of vertex it is). To resolve conflicts among multiple access requests to a memory block, the scheduler must employ some contention resolving policy. The solution for illustration here is FCFS (first come first served), that is, the earlier the arrival time of the packet, the higher the priority of the packet's memory access request.

In a trie-based routing scheme, a route searching request usually comprises multiple steps, one step per time slot for accessing a node stored in a memory block. Therefore, we express a route searching request as a tuple of the form $R(s_1, s_2..., s_i, ..., s_t)$ where $1 \leq i \leq t$ and $s_i \in \{m_1, m_2...m_M\}$. For instance, $R(m_1, m_2, m_5, m_4, m_3)$ denotes the list of memory blocks accessed in each step of the request $R(s_1, s_2, s_3, s_4, s_5)$, where m_1 for s_1, m_2 for s_2, m_5 for s_3 and so on. With the bipartite graph model, we give an algorithm framework for parallel scheduling in the following:

- Step 1: Each packet sends an access request to the memory block storing the trie's node that it is currently traversing to.
- Step 2: Each memory block arbitrarily chooses one from all its requesting packets to grant.
- Step 3: Each granted packet accesses the memory block it requested.

Scheduled in this way, the conflict-free memory accesses from multiple packets can be performed in parallel in a time slot, resulting in a pipeline system. Table I-II show a running example for the pipelined scheduling algorithm with the FCFS policy employed in step 2. Suppose that the number of memory blocks is $M = 4$, and the queue is initially empty, that is, $N = 0$. In Table I, we have 8 packets P_1-P_8 arriving at the beginnings of time slots 0-7, and these packets create the searching requests of R_1-R_8, respectively. Once a searching request is created, it will contend for the designated memory block in each scheduling instance until it is finished. For example, R_1 requests to access memory block m_1 in time slot 0. Because there is no other request, R_1 is granted and the maximum matching set (MMS) is $\{(R_1, m_1)\}$ for time slot 0, as given in the 1st row of Table II. In time slot 1, R_2 participates in the scheduling and requests m_1, while R_1 asks for m_2, resulting in the MMS of $\{(R_1, m_2), (R_2, m_1)\}$. In time slot 2, both R_1 and R_2 require accesses to m_3, hence a contention is caused and must be resolved. Governed by the FCFS principle for contention resolving, R_1 wins out the competition and the resulting MMS is $\{(R_1, m_3), (R_3, m_1)\}$. By further applying the scheduling algorithm to time slots 3-

Table I
INITIAL STATE AND ARRIVALS

Initial State($M = 4, N = 0$)		
Packet	Arrival Time Slot	Searching Request
P_1	0	$R_1(m_1, m_2, m_3, m_4)$
P_2	1	$R_2(m_1, m_3, m_3, m_2)$
P_3	2	$R_3(m_1, m_4, m_2, m_3)$
P_4	3	$R_4(m_1, m_4, m_1, m_2)$
P_5	4	$R_5(m_1, m_1, m_2, m_3)$
P_6	5	$R_6(m_1, m_2, m_4, m_1)$
P_7	6	$R_7(m_1, m_3, m_4, m_4)$
P_8	7	$R_8(m_1, m_2, m_3, m_1)$

Table II
PIPELINED SCHEDULING FOR THE ARRIVALS GIVEN IN TABLE I

Time Slot	MMS
0	$\{(R_1, m_1)\}$
1	$\{(R_1, m_2), (R_2, m_1)\}$
2	$\{(R_1, m_3), (R_3, m_1)\}$
3	$\{(R_1, m_4), (R_2, m_3), (R_4, m_1)\}$
4	$\{(R_2, m_3), (R_3, m_4), (R_5, m_1)\}$
5	$\{(R_2, m_2), (R_4, m_4), (R_5, m_1)\}$
6	$\{(R_3, m_2), (R_4, m_1)\}$
7	$\{(R_3, m_3), (R_4, m_2), (R_6, m_1)\}$
8	$\{(R_5, m_2), (R_7, m_1)\}$
9	$\{(R_5, m_3), (R_6, m_2), (R_8, m_1)\}$
10	$\{(R_6, m_4), (R_7, m_3), (R_8, m_2)\}$
11	$\{(R_6, m_1), (R_7, m_4), (R_8, m_3)\}$
12	$\{(R_7, m_4), (R_8, m_1)\}$

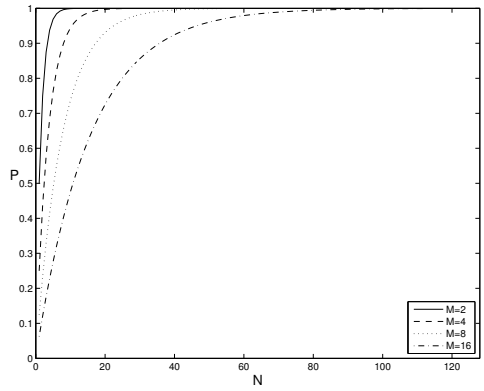


Fig. 3. $P(M, N)$ as a function of N, with $M \in \{2, 4, 8, 16\}$.

12, we get the corresponding results shown in Table II.

III. PRELIMINARY ANALYSIS OF QUEUING MODEL

The scheduler constructs and executes the MMS in each time slot, hence the pipeline efficiency as well as the system throughput is proportional to the average size of a MMS. To start the analysis, let $P(M, N)$ denote the probability for a memory block to get matched when there are N searching requests from the queuing packets. Because all the nodes of the prefix trie are evenly distributed into all the M memory blocks, in each time slot, the probability for a searching request to access a memory block is given by $\rho = \frac{1}{M}$. Hence we have $P(M, N)$ by:

$$P = \sum_{k=0}^{N-1} \binom{N}{k} \rho^{N-k} (1-\rho)^k = 1 - (1 - \frac{1}{M})^N \qquad \text{(III.1)}$$

Fig. 3 shows $P(M, N)$ as a function of N for $M \in \{2, 4, 8, 16\}$, respectively. From this figure, we see that for a given M, when N increases, P grows rapidly at the very beginning and gets steady when it is approaching the maximum, i.e. 1. This suggests that we can scale the system's aggregate throughput MP by increasing M and/or N.

IV. SIMULATION EXPERIMENTS

For performance evaluation of the proposed routing architecture under realistic traffic loads, we conduct a series of computer software simulations for the four trie-based routing algorithms BT, PT [8], 6-FST [9] and 2-MPT [10] running on the AS4637 BGP routing table [22], respectively. The nodes of the trie constructed by each algorithm are evenly distributed into all the memory blocks, i.e., the probability for a node to be stored into a memory block is $1/M$. The scheduling algorithm previously described in Section II-C is applied to pipeline the memory accesses for routing different packets. In this sense, the algorithms being investigated in our experiments are *pipelined* and are therefore referred as the pipelined BT, PT, 6-FST and 2-MPT algorithms, respectively.

A. Parameter Description

The performance measure to be investigated is the average queue length as a function of the normalized traffic load at the routing buffer. Each simulation result in the following presentation is the average of the samples from 10 runs, where each run consists of 20000 packet arrivals. For each newly arriving packet at the buffer, we randomly choose one from the 219581 entries of the AS4637 BGP routing table to assign as the packet's destination. The parameters used in the simulation experiments are summarized 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.
- Q: the average number of queuing packets in the routing buffer, at the beginning of a time slot (after the new arrival, if there is any).
- W: the normalized workload on a memory block, which is defined as $W = \lambda S/M$, where λ is the mean traffic arrival rate at the routing buffer.

B. Experiment Results

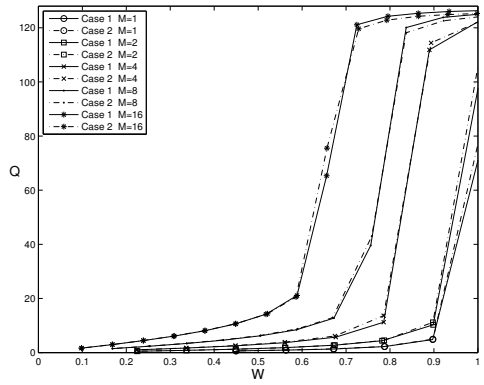
The experiments are conducted for two cases in respect to the traffic arrival processes at the routing buffer, i.e. the Bernoulli and Poisson processes, respectively.

- Case 1: Bernoulli process. The interval time between any two successive arrivals is Bernoulli distributed and each packet is of a fixed length of 1 time slot.
- Case 2: Poisson process. The interval time between any two successive arrivals is governed by an exponentially distribution and the packet length (in time slots) follows an exponential distribution as well.

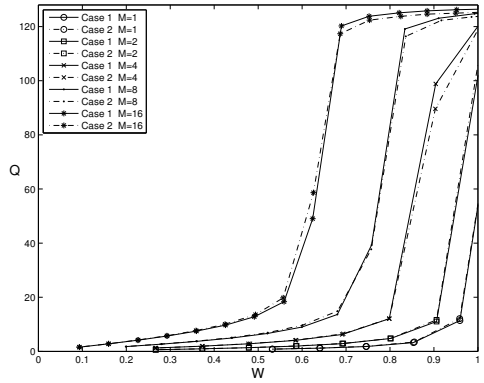
For these two cases, Fig. 4(a)-4(d) show the queue length Q as a function of the normalized workload W for the pipelined algorithms BT, PT, 6-FST, 2-MPT, with the buffer size of 128 packets and the number of memory blocks varying on 1, 2, 4, 8 and 16.

C. Simulation Analysis

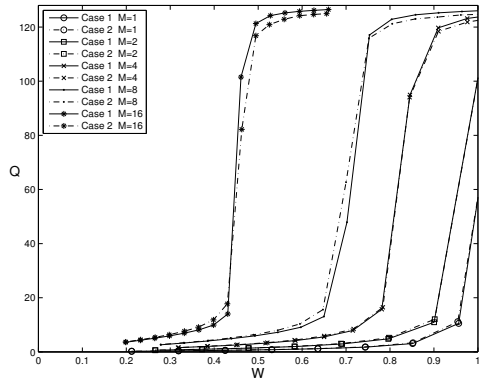
In Fig. 4(a)-4(d), each curve increases smoothly at the very beginning when the normalized workload W increases, and grows abruptly when the system is going to be overloaded,



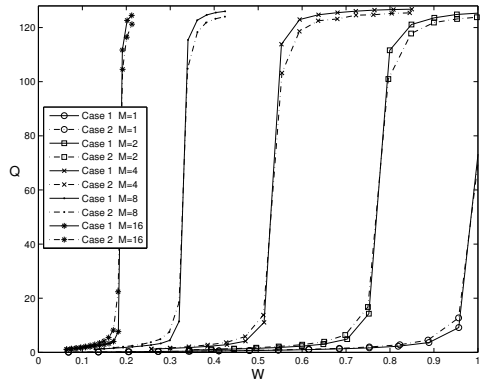
(a) Pipelined BT



(b) Pipelined PT



(c) Pipelined 2-MPT



(d) Pipelined 6-FST

Fig. 4. The average queue lengths given as functions of the normalized workloads, i.e. Q vs. W, for the pipelined BT, PT, 2-MPT and 6-FST algorithms, with $L = 128$, $R = 1$ and $M \in \{1, 2, 4, 8, 16\}$.

where W approaches a point termed as the threshold of W. It is observed that for each algorithm, the threshold of W gets smaller as the number of memory blocks M increases. For example, in Fig. 4(a), the threshold of W for the pipelined BT algorithm is decreased from around 0.9 to 0.8, 0.7, 0.6 when M is increased from 2 to 4, 8, 16, respectively. This can be explained as following. When M is small and the number of queuing packets are increased, it is likely that each memory will get matched with a packet. However, when M becomes larger, more queuing packets are needed for a memory to be matched with the same probability. Given a small buffer of 128 packets in our experiments, new arriving packets will be lost when the buffer is fully filled of queuing packets due to memory access conflicts. Consequently, for our experiment settings, the threshold of W is observed to be decreased when M is increased. Under the assumption that at most one packet per time slot can arrive at the routing buffer, which means $\lambda \leq 1$, W won't be able to reach 1 when $M > S$ (recalling that $W = \lambda S/M$). For example, in Fig. 4(c), $S = 10.62$ in this case, the maximum value of W is $S/M = 0.664$ for $M = 16$. Similarly in Fig. 4(d), $S = 3.42$ in this case, the maximum values of W for $M = 4, 8, 16$ are less than 1 too.

For a trie-based routing algorithm, a searching request must originate from the root node, resulting in that the root node is accessed far more frequently than the others. Consequently, the memory block storing the root node will be accessed more frequently, and likely see more access conflicts. This violates the assumption that the uniform probability of $1/M$ for a searching request to access each memory block, and may cause severe performance degradation. In our experiments, the values of S for BT, PT, 2-MPT, 6-FST are 22.46, 21.31, 10.62, 3.42, respectively. When S is smaller, the time for accessing the root node will play a critical role in the total time for completing a searching request. We observed in the experiments that the reason for the 6-FST algorithm behaved far more worse than the others is mainly due to access conflicts for the root node. This naturally suggests the following improvement scheme to store multiple copies of the root node for better performance.

D. Improvement Using Root Node Copies

The analysis for the simulation results in section IV-C indicates that the memory access conflicts for the root node impose a severe negative impact on the system's achievable throughput. To overcome this problem, we can store multiple copies of the root node in different memory blocks, one copy per memory block, to reduce contentions. Let R denote the number of copies for the root node, where $1 \leq R \leq M$. Without loss of generality, these copies are stored in the memory blocks $m_1, m_2, ..., m_R$, respectively. For a searching request trying to access the root node, it can be done if any of these copies is available. In other words, the root node can not be accessed only when all these R memory blocks are occupied by other searching requests. This will yield an increased success probability for accessing the root node. Because in this study, the contentions for each memory block are resolved by the FCFS principle, the scheduling result at

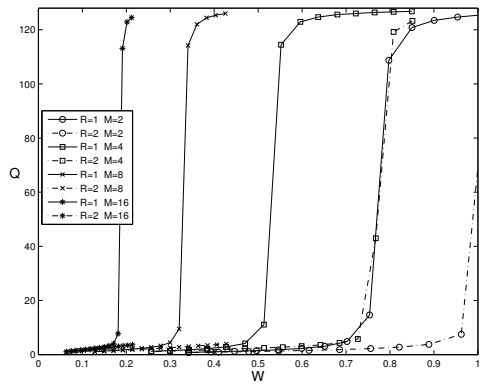


Fig. 5. Q vs. W for the pipelined 6-FST algorithm, with $L = 128$, $M \in \{2, 4, 8, 16\}$ and $R \in \{1, 2\}$.

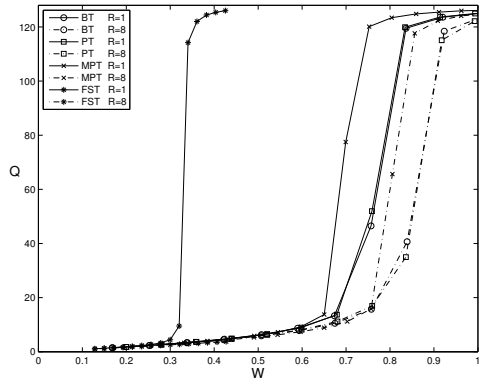


Fig. 6. Q vs. W for the pipelined BT, PT, 2-MPT and 6-FST algorithms, with $L = 128$, $M = 8$ and $R \in \{1, 8\}$.

each time slot is therefore deterministic and unique. Fig. 5 shows the performance of 6-FST with $R \in \{1, 2\}$, from which we see a substantial improvement on the threshold of W is achieved by using only one more copy of the root node.

To get more insights into the effects of R on the threshold of W, we show in Fig. 6 a performance comparison for the pipelined algorithms BT, PT, 2-MPT and 6-FST with $M = 8$ and $R \in \{1, 8\}$. From this figure, we see that the threshold of W gets increased by using more root node copies for all the investigated algorithms.

V. Conclusion

We have proposed a scalable routing architecture to support the pipeline running of any trie-based IP address routing algorithm that links prefix nodes via pointers. To scale the routing system's throughput, we use multiple memory blocks to increase the aggregate bandwidth of the memory system for faster accesses of the trie's nodes. In addition, different from the existing routing architectures with deterministic routing delays, we employ a routing buffer to tolerate temporarily access contentions on each individual memory block. The results from both the analytical queuing model and the simulation experiments indicate that this architecture can be a potential candidate for building high bandwidth routing engines. To further elaborate this study, we are currently investigating various scheduling algorithms to pipeline the memory accesses of queuing packets, as well as a feasible design for implementing the proposed architecture.

VI. Acknowledgment

This research work was supported in part by the National Natural Science Foundation of China Grant No. 60873056 and the Fundamental Research Funds for the Central Universities of China Grant No. 11lgzd04.

References

[1] K. Kim and S. Sahni, "Efficient construction of pipelined multibit-trie router-tables," *IEEE Trans. on Computers*, pp. 32–43, Jan 2007.
[2] A. Basu and G. Narlikar, "Fast incremental updates for pipelined forwarding engines," *IEEE/ACM Trans. on Networking*, pp. 690–703, Jun 2005.
[3] 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.
[4] 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.
[5] W. Lu and S. Sahni, "Packet forwarding using pipelined multibit tries," in *Proc. IEEE Symp. on Computers and Communications*, June 2006, pp. 26–29.
[6] F. aboescu, 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.
[7] 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.
[8] M. Berger, "IP lookup with low memory requirement and fast update," in *Proc. IEEE HPSR*, Jun 2003, pp. 287–291.
[9] S. Sahni and K. Kim, "Efficient construction of multibit tries for IP lookup," *IEEE/ACM Trans. on Networking*, pp. 650–662, Aug 2003.
[10] 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.
[11] V. Srinivasan and G. Varghese, "Faster IP lookups using controlled prefix expansion," *ACM Trans. on Computer Systems*, pp. 1–40, Feb 1999.
[12] S. Nilsson and G. Karlsson, "IP-address lookup using lc-tries," *IEEE J. on Selected Areas in Communications*, pp. 1083–1092, Jun 1999.
[13] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small forwarding tables for fast routing lookups," in *Proc. ACM SIGCOMM*, 1997, pp. 3–14.
[14] 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.
[15] ——, "Efficient construction of variable-stride multibit tries for IP lookup," in *Proc. Symp. on Applications and the Internet*, Feb 2002, pp. 220–227.
[16] 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.
[17] H. Lim, C. Yim, and E. Swartzlander, "Priority tries for IP address lookup," *IEEE Trans. on Computers*, pp. 784–794, June 2010.
[18] W. Lu and S. Sahni, "Recursively partitioned static IP router-tables," *IEEE Trans. on Computers*, pp. 1683–1690, Dec 2010.
[19] M. Sanchez, E. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," *IEEE Network*, pp. 8–23, 2001.
[20] S. Sahni, K. Kim, and H. Lu, "Data structures for one-dimensional packet classification using most-specific-rule matching," *Int. J. of Foundations of Computer Science*, pp. 337–358, 2003.
[21] G. Huston, *Analyzing the Internet's BGP routing table*, July 2003.
[22] "BGP routing table analysis reports," 2007.