

Towards Green Routers: Depth-Bounded Multi-Pipeline Architecture for Power-Efficient IP Lookup *

Weirong Jiang and Viktor K. Prasanna
Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089, USA
{weirongj, prasanna}@usc.edu

Abstract

Power consumption has become a major concern in designing IP lookup engines for next generation routers. Although TCAMs dominate today's high-end routers, they are not scalable in terms of clock rate and power consumption. SRAM-based pipeline solutions are considered promising alternatives for high-speed IP lookup engines. However, existing SRAM-based pipeline architectures suffer from high power consumption in the worst cases, due to the large memory size and the long pipeline depth. This paper proposes a power-efficient SRAM-based pipelined IP lookup engine for future "green" routers. Both chip-level parallelism and clock gating techniques are employed to reduce the power consumption. With the aid of small TCAMs, a two-phase scheme is proposed to partition a routing trie into a number of height-bounded subtrees, which are then mapped onto multiple pipelines. Each IP lookup is completed through a bounded number of accesses on small size memories. Simulation experiments using real-life traces show that our solution can store a backbone routing table with over 200K prefixes in 4.25 MB memory, sustains a throughput of 400 Gbps, and achieves up to 7-fold and 3-fold reductions in power consumption over the state-of-the-art TCAM-based and SRAM-based solutions, respectively.

1 Introduction

The primary function of network routers is to forward packets based on the lookup results from matching the destination IP address of the packet to the entries in the routing table. As link rates increase beyond OC-768 rate (i.e. 40 Gbps), IP lookup becomes a major performance bottleneck

*This work is supported by the United States National Science Foundation under grant No. CCF-0702784.

for network routers [5, 7]. Furthermore, as routers achieve aggregate throughputs of trillions of bits per second, power consumption by lookup engines becomes an increasingly critical concern for backbone router design [6, 14].

Ternary Content Addressable Memories (TCAMs), where a single clock cycle is sufficient to perform one IP lookup, dominate today's high-end routers. However, as a result of the massive parallelism inherent in their architecture, TCAMs do not scale well in terms of clock rate, power consumption, or chip density [7]. Taylor [14] estimates that the power consumption per bit of TCAMs is on the order of 3 micro-Watts, which is 150 times more than for Static Random Access Memories (SRAMs).

On the other hand, most SRAM-based algorithmic solutions can be implemented as some form of tree traversal, such as trie-based algorithms [13], which can be pipelined to achieve a high throughput of one packet per clock cycle [3, 7]. Though SRAM-based pipeline solutions are being considered as attractive alternatives to TCAMs for next-generation routers [7], most SRAM-based solutions still suffer from high power consumption [15], which comes mainly from two sources: First, the size of the memories to store the search structure is large. Second, the number of accesses to these large memories (i.e. the pipeline depth), is large. The overall power consumption for one IP lookup can be expressed as in Equation 1:

$$Power_{overall} = \sum_{i=1}^H [P_m(S_i, \dots) + P_l(i)] \quad (1)$$

Here, H denotes the number of memory accesses, i.e. the pipeline depth, $P_m(\cdot)$ the function of the power dissipation of a memory access (which usually has a positive correlation with the memory size), S_i the size of the i th memory being accessed, and $P_l(i)$ the power consumption of the logic associated with the i th memory access. Since the logic dissipates much less power than the memories in the memory-dominant architectures [10, 11], the main fo-

cus of this paper is on reducing the power consumption of the memory accesses. Note that the power consumption of a single memory is affected by many other factors, such as the fabrication technology and sub-bank organization, which are beyond the scope of this paper. According to Equation 1, to reduce the worst-case power consumption, we should bound the number of memory accesses, as well as minimize the memory size for each access.

As shown in [15], the power consumption of TCAM-based IP lookup engines can be dramatically reduced by partitioning the routing table and mapping onto multiple TCAM blocks. The power consumption of such blocked TCAMs is even comparable to that of DRAM/SRAM-based solutions. We believe that SRAM-based pipeline solutions can also achieve a lower order of power consumption than those “cool” TCAMs [15], by partitioning the routing table in a different but efficient way.

Different IP addresses may have various numbers of memory accesses and different sequences of accessed memories. The number of memory accesses for each IP address depends on the height of the tree the IP is traversing. Thus, it is necessary to bound the tree height for all IP addresses while minimizing the overhead.

This paper addresses the above problems and makes the following contributions.

- We explore an idea similar to that in [15], to exploit chip-level parallelism by partitioning the routing table and allowing IP lookup be performed on only one of the partitions. We propose a SRAM-based multi-pipeline architecture, shown in Figure 1. In contrast to [15], which aims to allocate equal numbers of prefix entries to each TCAM block, we map the partitioned routing table onto multiple SRAM-based pipelines to ensure each pipeline requires the same amount of memory.
- Based on the observations of the prefix length distribution in real-life routing tables, we propose a two-phase partitioning scheme, including an effective method called *height-bounded split*, to partition a routing trie into several height-bounded subtrees. As a result, all the IPs traverse the subtrees in a bounded number of memory accesses. The partitioning scheme is enabled by using small TCAMs as part of the index table. To the best of our knowledge, this is among the first works on TCAM/SRAM hybrid architectures.
- Within each pipeline, the memory distribution over stages should be balanced so that the largest stage is not the performance bottleneck of the pipeline. We revise the node-to-stage mapping scheme from our previous work [7], which allows the trie nodes on the same level to be mapped onto different stages. By exploiting the next-hop information, the number of leaf

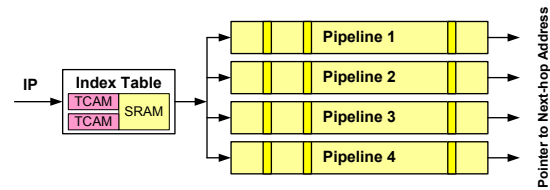


Figure 1. Partitioning-based multi-pipeline architecture ($D = 4$).

nodes for mapping is dramatically reduced, and thus memory efficiency is achieved. Furthermore, clock gating is employed to eliminate the power consumption in the skipped pipeline stages.

- Simulation experiments using real-life backbone routing tables demonstrate the SRAM-based multi-pipeline architecture to be a promising solution for future green routers. The proposed 4-pipeline architecture stores a backbone routing table with over 200K prefixes in 4.25 MB memory, sustains a high throughput of 400 Gbps for minimum size (40 bytes) packets, and achieves up to 7-fold and 3-fold reductions in power consumption over the state-of-the-art TCAM-based and SRAM-based solutions, respectively.

The rest of the paper is organized as follows. Section 2 reviews the background and related work. Section 3 proposes the algorithms to partition and map a routing table onto the multi-pipeline architecture. Section 4 discusses the issues of hardware implementation. Section 5 presents experiments with real-life backbone routing tables to evaluate the performance. Section 6 concludes the paper.

2 Background and Related Work

2.1 Trie-based IP Lookup

The entries in the routing tables are specified using prefixes, and the nature of IP lookup is longest prefix matching (LPM). The most common data structure in algorithmic solutions for LPM is some form of trie [13]. A trie is a binary tree, where a prefix is represented by a node. The value of the prefix corresponds to the path from the root of the tree to the node representing the prefix. The branching decisions are made based on the consecutive bits in the prefix. A trie is called a uni-bit trie if only one bit at a time is used to make branching decisions. Figure 2 (b) shows the uni-bit trie for the prefix entries in Figure 2 (a). For example, the prefix “011*” corresponds to the path starting at the root and ending in node P6: first a left-turn (0), then a right-turn (1), and finally a turn to the right (1). Each trie node contains two fields: the represented prefix and the pointer to the child nodes.

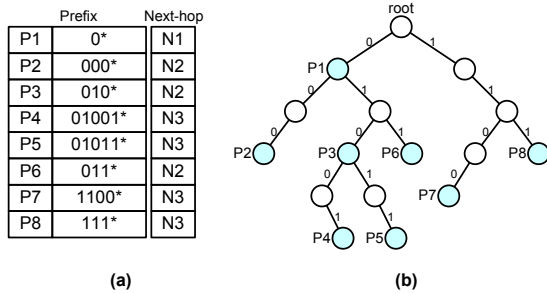


Figure 2. (a) Prefix entries; (b) Trie.

Given a uni-bit trie, IP lookup is performed by traversing the trie according to the bits in the IP address. The information about the longest prefix matched so far, which is updated when meeting a node containing a prefix, is carried along the traversal. Thus, when a leaf is reached, the longest matched prefix along the traversed path is returned. The time to look up a uni-bit trie is equal to the prefix length. The use of multiple bits in one scan can increase the search speed. Such a trie is called a multi-bit trie. The number of bits scanned at a time is called the *stride*. Multi-bit tries using a larger stride usually result in much larger memory consumption, though some optimization schemes have been proposed to reduce memory wastage [5]. For simplicity, we consider only the uni-bit trie in this paper, though our ideas can be applied to other types of trie.

2.2 Partitioning the Routing Table

To reduce the power consumption of TCAMs, Zane et al. [15] propose two methods to partition a routing table into several blocks and perform IP lookup on one of the blocks. The aim is to ensure each block contains equal numbers of prefixes, so that the power consumption reduction is maximized. The first method is called *bit-selection*, where selected bits from the input are used to index different blocks directly. However, prefix distribution imbalance among the blocks may be quite high, resulting in low worst-case performance. The second method is called the *trie-based* approach, which carves subtrees out of the full trie. This approach can have a much better worst-case bound, while it needs an index TCAM and an index SRAM to implement the subtree-to-block mapping.

In contrast to TCAMs where each prefix is stored as one word, the memory size in trie-based solutions is proportional to the number of trie nodes. Hence, the aim of routing table partitioning in the multi-pipeline architecture is to ensure each pipeline contains equal numbers of trie nodes rather than of prefixes. Our previous work [7] simply partitions the routing trie into many subtrees of various sizes, and then uses an approximation algorithm to map those subtrees to multiple pipelines. As a result, each pipeline contains ap-

proximately equal numbers of nodes. However, the smallest and largest pipeline sizes vary by as much as a factor of 1.5.

2.3 Memory-Balanced Pipelines

Pipelining can dramatically improve the throughput of trie-based solutions. A straightforward way to pipeline a trie is to assign each trie level to a different stage, so that a lookup request can be issued every clock cycle. However, this results in unbalanced memory distribution across the pipeline stages, which has been identified as a dominant issue for SRAM-based pipeline architectures [2]. In an unbalanced pipeline, more time is needed to access the larger local memory. This leads to a reduction in the global clock rate. Furthermore, since it is unclear at hardware design time which stage will be the fattest, we must allocate memory with the maximum size for each stage. Such an over-provisioning results in memory wastage and excessive power consumption [2].

Basu et al. [3] and Kim et al. [9] both reduce the memory imbalance using variable strides, which is difficult to implement in hardware. Baboescu et al. [2] and Kumar et al. [10] propose two non-linear pipeline architectures, named Ring and CAMP, to achieve a balanced memory distribution over stages. However, due to their non-linear structures, neither Ring nor CAMP can sustain a throughput of one packet per clock cycle, or support the *write bubble* proposed in [3] for the incremental route update. Our previous work [7] proposes a heuristic to perform fine-grained node-to-stage mapping to achieve balanced memory distribution across pipeline stages. This paper optimizes the mapping algorithm to achieve memory efficiency, based on the observation that the number of next-hop information is much less than that of prefixes.

2.4 Existing Power-Efficient SRAM-based Solutions

Kaxiras et al. [8] propose a SRAM-based approach called IPStash for power-efficient IP lookup. IPStash replaces the full associativity of TCAMs with set associative SRAMs to reduce power consumption. However, the set associativity depends on the routing table size and thus is not scalable. For large-scale routing tables, the set associativity is still large, resulting low clock rate and high power consumption.

Peng et al. [11] propose a scheme to cache the supernodes of the trie in order to skip the access to the deeper trie levels. The average number of memory accesses for looking up an IP address is reduced, and thus the average power consumption decreases. However, the caching-based schemes achieve only good statistical performance, and do not improve the power consumption in the worst cases.

3 Algorithms

First, we define the following terms.

Definition 1 The depth of a trie node is the directed distance from the trie node to the trie root. The depth of a trie refers to the maximum depth of all trie leaves.

Definition 2 The height of a trie node is the maximum directed distance from the trie node to a leaf node. The height of a trie refers to the height of the root. In fact, the depth of a trie is equal to its height.

Definition 3 The size of a trie is the number of nodes in the trie.

In the worst case, the number of memory accesses needed for an IP lookup is equal to the trie height. We propose a holistic scheme to (1) partition a full routing trie into many height-bounded subtrees, and (2) map those subtrees onto multiple pipelines so that each pipeline contains equal numbers of trie nodes. By these means, each IP lookup is completed through a bounded number of accesses on small-size memories, so that power efficiency is achieved according to Equation 1.

Our scheme consists of two phases: prefix expansion and height-bounded split, as illustrated in Figure 3 (a) and (b).

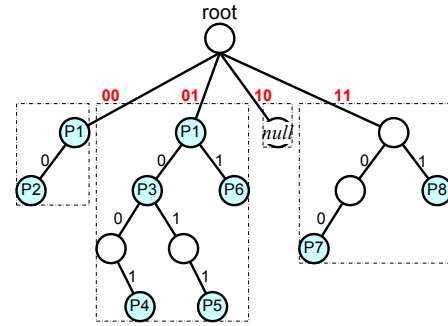
3.1 Prefix Expansion

Several initial bits are used as the index to partition the trie into many disjoint subtrees. The number of initial bits to be used is called the *initial stride*, denoted I . A larger I can result in more subtrees of smaller height, which can help balance the memory distribution among pipelines as well as across stages when mapping subtrees to pipelines. However, a large I can result in size (prefix) expansion, where the sum of the sizes (number of prefixes) of all subtrees is larger than the size (number of prefixes) of the original trie.

Table 1. Representative Routing Tables

| Routing table | Location | # of prefixes |
|---------------|------------------------|---------------|
| RIPE-NCC | Amsterdam, Netherlands | 243474 |
| LINX | London, UK | 240797 |
| SFINX | Paris, France | 238089 |
| NYIIX | New York, USA | 238836 |
| DE-CIX | Frankfurt, Germany | 243732 |
| MSK-IX | Moscow, Russia | 238461 |
| PAIX | Palo Alto, USA | 243731 |
| PTTMetro-SP | Sao Paulo, Brazil | 243242 |

(a) Prefix Expansion



(b) Height-Bounded Split ($B_H = 3$)

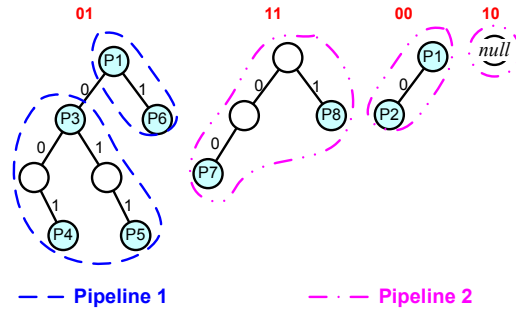


Figure 3. Partition the trie shown in Figure 2 (b) and map onto 2 pipelines. The value of the height bound is $B_H = 3$.

We studied the prefix length distribution based on 8 backbone routing tables collected from [12] on 11/30/2007. Their information is listed in Table 1. We obtained results similar to the findings of other researchers [8]: the majority of prefixes has lengths between 16 and 24. Since few prefixes are shorter than 16, $I < 16$ should not result in much size expansion. To verify this, we conducted experiments using various values of I on the above 8 routing tables, and examined the size expansion factor (SEF) and the prefix expansion factor (PEF), which are defined in Equations (2) and (3), respectively.

$$SEF = \frac{\sum_{all\ subtrees} Size(the\ subtree)}{Size(the\ original\ trie)} \quad (2)$$

$$PEF = \frac{\sum_{all\ subtrees} PrefixCount(the\ subtree)}{PrefixCount(the\ original\ trie)} \quad (3)$$

The experimental results are shown in Figure 4. We found that, when $I = 16$, the size expansion factor was less than 1.04, while the prefix expansion factor was below 1.25. In the following sections, we pick $I = 12$ as a default, which resulted in little size or prefix expansion.

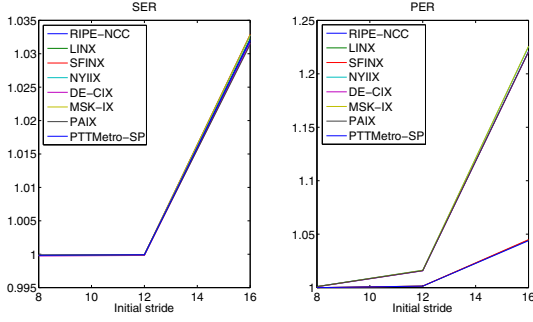


Figure 4. Experimental results for the 8 routing tables: SER and PER

3.2 Height-Bounded Split

After prefix expansion, we obtain $K = 2^I$ subtries, whose heights can vary from 1 to $32 - I$. We must partition these subtries further, to reduce the height of the resultant subtries, with minimum overhead. Meanwhile, we must map those subtries onto multiple pipelines to achieve a balanced memory allocation among those pipelines. In other words, given D pipelines (denoted P_i , $i = 1, \dots, D$) and K subtries each of which has W_i nodes ($i = 1, 2, \dots, K$), we must ensure that each pipeline (except the last pipeline) contains $B_P = \left\lceil \frac{\sum_{i=1}^K W_i}{D} \right\rceil$ nodes.

The above problem is a variant of packing problems, and can be proved to be NP-hard. Using an approximation algorithm, we first sort those subtries obtained from prefix expansion, in decreasing order of size. Then we traverse those subtries one by one. Each subtrie is traversed in the post-order. Once the height bound, denoted B_H , is reached, a new subtrie is split. After the number of nodes mapped onto a pipeline exceeds the size bound of the pipeline, B_P , we map the rest of nodes onto the next pipeline. Algorithm 1 shows a recursive implementation of the height-bounded split, where $size(P_i)$ denotes the number of nodes mapped onto the i th pipeline.

3.3 Node-to-Stage Mapping

We now have a set of height-bounded subtries for each pipeline. Within each pipeline, the trie nodes should be mapped to the stages, while keeping the memory requirement across stages balanced. In other words, given H stages and K' subtries, each of which has W'_i nodes ($i = 1, 2, \dots, K'$), each stage should contain $B_G = \left\lceil \frac{\sum_{i=1}^{K'} W'_i}{H} \right\rceil$ nodes. Also, each pipeline should be linear. To perform the

Algorithm 1 Height-bounded split: $HBS(n, i)$

Input: n : a node;

Input: i : the ID of the pipeline for n to be mapped on.

Output: i : the ID of the pipeline for the next node to be mapped on.

- 1: **if** $n == null$ **then**
 - 2: Return i .
 - 3: **end if**
 - 4: $i = HBS(n.left_child, i)$
 - 5: $i = HBS(n.right_child, i)$
 - 6: **if** $size(P_i) < B_P$ **then**
 - 7: Map n onto P_i .
 - 8: **else**
 - 9: Map n onto P_{i+1} .
 - 10: **end if**
 - 11: **if** $n.height \geq B_H$ **then**
 - 12: Mark n as a subtrie root.
 - 13: **end if**
 - 14: Return $i + 1$.
-

node-to-stage mapping, we use a heuristic similar to that in [7], by allowing the nodes on the same level of a subtrie to be mapped onto different stages. There is only one constraint:

Constraint 1: If node A is an ancestor of node B in a trie, then A must be mapped to a stage preceding the stage to which B is mapped.

3.3.1 Leaf Reduction

We note that there are leaf nodes sharing the same next-hop information. For example, in Figure 5(a), there are two identical leaf nodes, both of which contain P3. Thus, we propose an optimization called *leaf reduction*, shown in Figure 5(b). We performed experiments on the 8 backbone routing tables described in Section 3.1. Assuming there were 64 next-hop ports in the router, we found that over 99% of the leaf nodes could be removed after the optimization, resulting in up to 35% less memory used.

3.3.2 Mapping Algorithm

After leaf reduction, we map the trie nodes onto pipeline stages using Algorithm 2. R_n denotes the number of remaining nodes to be mapped onto stages, and R_h the number of remaining stages onto which nodes can be mapped. $Child(n)$ denotes the child nodes of node n . A *CriticalNode* is a node whose height is larger than the number of remaining stages. If such a node is not mapped onto the current stage, some of its descendants can not be mapped later. Since a leaf node may have multiple parents as a result of leaf reduction, the leaf node cannot be mapped onto the stages until all parents of the node have

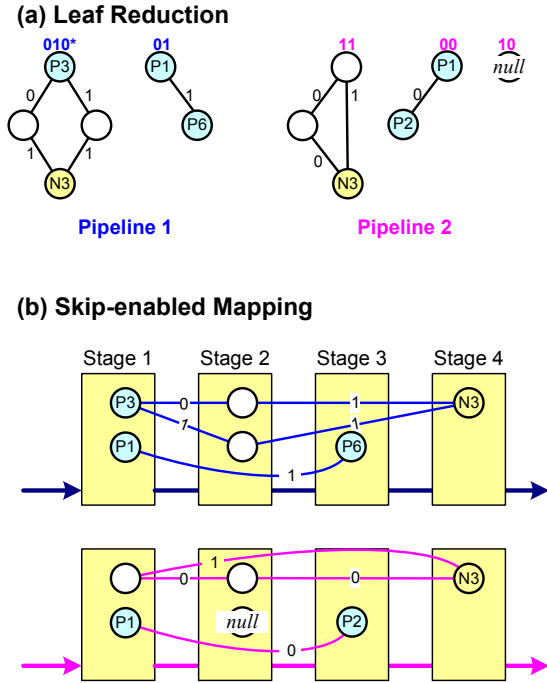


Figure 5. Node-to-stage mapping for the subtrees shown in Figure 3 ($H = 4$).

been mapped. The complexity of this algorithm is $O(HN)$, where H denotes the pipeline depth, and N the total number of trie nodes.

4 Implementation Issues

This section discusses several key issues of hardware implementation of the proposed architecture.

4.1 Index Table

After height-bounded splitting, the resulting subtrees may be rooted at different depths of the original trie. For the subtrees rooted at the depth of I , we use an index SRAM (called SRAM_A). For the rest subtrees, we need an index TCAM and an index SRAM (called SRAM_B). The TCAM stores the prefixes to represent the subtrees. The two index SRAMs store the information associated with each subtree: (1) the mapped pipeline ID, (2) the ID of the stage where the subtree's root is stored, and (3) the address of the subtree's root in that stage. Figure 6 shows the index table used to achieve the mapping shown in Figure 3.

An arriving input IP searches the index SRAM_A and the index TCAM in parallel. I initial bits of the input IP are used to index the SRAM_A. Meanwhile, the entire input IP

Algorithm 2 Node-to-stage mapping

Input: L subtrees: $\{T_i | i = 1, 2, \dots, L\}$; and H empty stages.

Output: H stages with mapped nodes.

- 1: Initialization: $ReadyList = \phi$, $NextReadyList = \phi$, $R_n = \sum_{i=1,2,\dots,L} size(T_i)$, $R_h = H$.
- 2: Map the roots of the subtrees into Stage 1. Push the children of the mapped nodes into $ReadyList$.
- 3: $R_n = R_n - M_1$, $R_h = R_h - 1$.
- 4: **for** $i = 2$ to H **do**
- 5: Sort the nodes in $ReadyList$ in the decreasing order of their heights.
- 6: **while** ($M_i < R_n/R_h$ AND $Readylist \neq \phi$) OR $\exists CriticalNode$ **do**
- 7: Pop the node from $ReadyList$. Map the popped node n onto Stage i . $M_i = M_i + 1$.
- 8: **if** All the parents of $Child(n)$ have been mapped **then**
- 9: Push $Child(n)$ into $NextReadyList$.
- 10: **end if**
- 11: **end while**
- 12: $R_n = R_n - M_i$, $R_h = R_h - 1$.
- 13: Merge the $NextReadyList$ to the $ReadyList$.
- 14: **end for**

searches the index TCAM and obtains the subtree ID corresponding to the longest matched prefix. Then, the IP uses the subtree ID to index the SRAM_B to retrieve the associated information. The result obtained from the SRAM_B has a higher priority than that from the SRAM_A. The number of entries in the SRAM_A is 2^I , while the number of entries in the index TCAM and SRAM_B is at most 2^{32-B_H} .

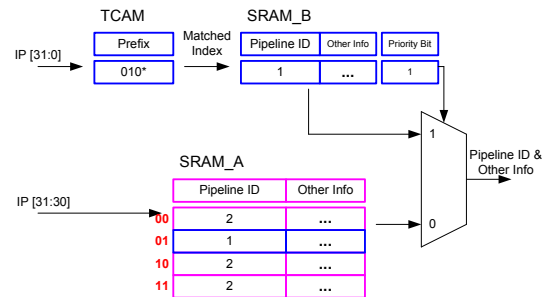


Figure 6. Index table for the mapping shown in Figure 3

4.2 Pipeline Structure

As shown in Figure 1, there are multiple linear pipelines. All pipelines have the same depth, which is determined by

the height bound. If the height bound is B_H , the pipeline depth must be $H \geq B_H$. In the following sections, $H = B_H + 1$.

Inside each pipeline, we use a method similar to that in our previous work [7], to allow two nodes on the same subtree level to be mapped to different stages. Each node stored in the local memory of a pipeline stage has three fields: (1) the prefix or the next-hop pointer, (2) the memory address of its child node in the pipeline stage where the child node is stored, and (3) the distance to the pipeline stage where the child node is stored. When a packet is passed through the pipeline, the distance value is decremented by 1 when it goes through a stage. If the distance value becomes 0, the child node's address is used to access the memory in that stage. Otherwise, the clock signal is gated with the read enable signal to eliminate the power consumption for the current stage.

We update the memory in the pipeline by inserting *write bubbles* [3]. Since the pipeline is linear, all packets preceding or following the write bubble can perform their IP lookup while the write bubble performs an update. Updating one route needs only one write bubble, which can update multiple stages during one traversal.

5 Performance Evaluation

To evaluate the proposed schemes, we conducted simulation experiments on the 8 real-life backbone routing tables described in Section 3. Considering ASIC implementation, we used CACTI 5.3 [4] and an accurate TCAM model [1] to evaluate the performance of SRAMs and TCAMs, respectively. The number of pipelines was $D = 4$, and the initial stride for prefix expansion was $I = 12$.

5.1 Selection of Height Bound

How to set an appropriate height bound is an issue worth discussing. Using a small height bound can reduce the number of memory accesses for IP lookup, but it may also result in a large number of subtrees, which requires a large table to index those subtrees. A subtree whose height is h may be split into $O(2^{h-B_H})$ subtrees, whose heights are bounded by B_H .

We conducted experiments with various values of the height bound to evaluate such a trade-off. As Figure 7 shows, a smaller height bound resulted in a larger number of TCAM entries in the index table. The architecture achieved the lowest power consumption¹ when the value of the height bound was 16. In the following experiments, the height bound was $B_H = 16$, and the pipeline depth was $H = B_H + 1 = 17$.

¹The measurement of the power consumption was similar as in Section 5.2.

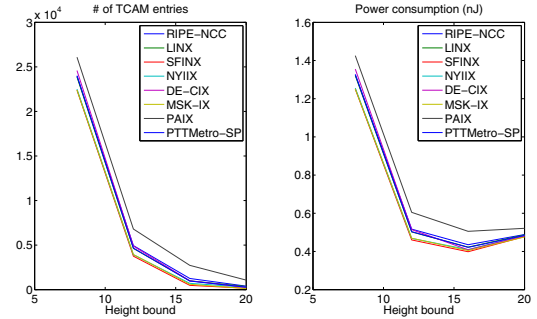


Figure 7. Impact of height bound

5.2 Architecture Performance

We mapped the 8 routing tables onto a 4-pipeline 17-stage architecture. As Figure 8 shows, our partitioning and mapping scheme achieved a balanced memory allocation among the 4 pipelines.

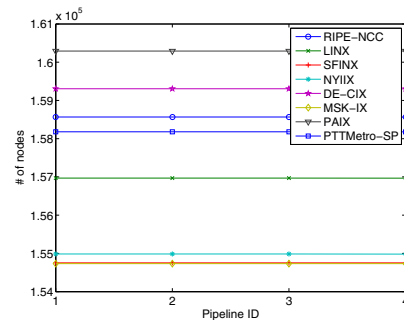


Figure 8. Node distribution over 4 pipelines.

The node distribution over stages within each pipeline is shown in Figure 9. The first several stages of a pipeline could not be balanced, since there were few nodes at the top levels of any subtree. According to Figure 9, each stage contained fewer than 16K nodes. Thus 14 address bits were enough to index a node in the local memory of a stage. The pipeline depth was 17, and the first stage was dedicated for the subtree roots. Thus we needed 4 bits to specify the distance for each node. Assuming each node needed 15 bits as the pointer to next-hop information, the total memory needed to store 243732 prefixes from the largest routing table DE-CIX in this architecture was $(13 + 4 + 15) \times 2^{14} \times 17 \times 4 \approx 34 \text{ Mb} = 4.25 \text{ MB}$, where each stage needed 64 KB of memory. According to CACTI 5.3 [4], a 64 KB SRAM using 65 nm technology needed 0.8017 ns to access and dissipated 0.0235 nJ power. For DE-CIX, there were 986 entries in the index TCAM. According to the TCAM model [1], a 986-row 18-bit TCAM

Table 2. Comparison on power consumption (nJ) of different solutions

| Routing table | RIPE-NCC | LINX | SFINX | NYIIX | DE-CIX | MSK-IX | PAIX | PTTMetro-SP |
|--------------------------------|----------|--------|--------|--------|--------|--------|--------|-------------|
| 4-partition CoolCAM [15] | 2.8881 | 2.8571 | 2.8246 | 2.8337 | 2.8920 | 2.8286 | 2.8920 | 2.8856 |
| IPStash [8] (normalized) | 1.1553 | 1.1428 | 1.1298 | 1.1335 | 1.1568 | 1.1314 | 1.1568 | 1.1543 |
| 4-pipeline arch ($B_H = 16$) | 0.4353 | 0.4227 | 0.3986 | 0.4095 | 0.4410 | 0.4050 | 0.5055 | 0.4215 |

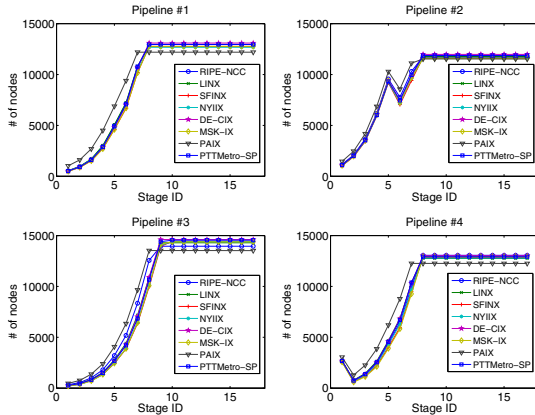


Figure 9. Node distribution over 17 stages in each pipeline.

using 65 nm technology needed 1.4653 ns to access and dissipated 0.0325 nJ power. The overall clock rate could achieve $\frac{1}{0.8017} = 1.25$ GHz, by using two copies of the index TCAM working in parallel. The throughput was thus 400 Gbps (i.e. $10 \times$ OC-768 rate) for minimum size (40 bytes) packets. The power consumption for one IP lookup was $0.0235 \times 16 + 0.0325 \times 2 = 0.441$ nJ².

The results for the 8 routing tables are compared with the 4-partition CoolCAM [15] and IPStash [8] in Table 2. The CoolCAM results did not include the power dissipation of the index TCAM. The IPStash results were normalized using the best-case reduction ratio given in [8]. Our architecture achieved up to 7-fold and 3-fold reductions in power consumption over CoolCAMs and IPStash, respectively.

6 Conclusions

TCAMs do not scale well in terms of power consumption, and most SRAM-based solutions suffer from high worst-case power consumption due to the unbounded number of accesses on large size memories. This paper shows a paradigm where small TCAMs are combined with a SRAM-based architecture to achieve power efficiency. A two-phase scheme is proposed to partition a routing trie into

²The two index SRAMs, one with $2^{10} \times 10$ bits = 1.25 KB and the other with $2^{12} \times 18$ bits = 9 KB, were so small that their contribution to the overall performance could be ignored.

many height-bounded subtrees and map them onto multiple pipelines, while keeping the memory requirement among pipelines balanced. Our experiments using real-life traces show that the proposed 4-pipeline architecture achieves up to 7-fold and 3-fold reductions in power consumption over the state-of-the-art TCAM-based and SRAM-based solutions, respectively, while sustaining a throughput of 400 Gbps (i.e. $10 \times$ OC-768 rate) for backbone routing tables.

References

- [1] B. Agrawal and T. Sherwood. Ternary CAM power and delay model: Extensions and uses. *IEEE Trans. VLSI Syst.*, 16(5):554–564, 2008.
- [2] F. Baboescu, D. M. Tullsen, G. Rosu, and S. Singh. A tree based router search engine architecture with single port memories. In *Proc. ISCA*, pages 123–133, 2005.
- [3] A. Basu and G. Narlikar. Fast incremental updates for pipelined forwarding engines. In *Proc. INFOCOM*, pages 64–74, 2003.
- [4] CACTI 5.3. <http://quid.hpl.hp.com:9081/cacti/>.
- [5] W. Eatherton, G. Varghese, and Z. Dittia. Tree bitmap: hardware/software IP lookups with incremental updates. *SIGCOMM Comput. Commun. Rev.*, 34(2):97–122, 2004.
- [6] M. Gupta and S. Singh. Greening of the Internet. In *Proc. SIGCOMM '03*, pages 19–26, 2003.
- [7] W. Jiang, Q. Wang, and V. K. Prasanna. Beyond TCAMs: An SRAM-based parallel multi-pipeline architecture for terabit IP lookup. In *Proc. INFOCOM*, 2008.
- [8] S. Kaxiras and G. Keramidas. IPStash: a set-associative memory approach for efficient IP-lookup. In *INFOCOM*, pages 992–1001, 2005.
- [9] K. S. Kim and S. Sahn. Efficient construction of pipelined multibit-trie router-tables. *IEEE Trans. Comput.*, 56(1):32–43, 2007.
- [10] S. Kumar, M. Becchi, P. Crowley, and J. Turner. CAMP: fast and efficient IP lookup architecture. In *Proc. ANCS*, pages 51–60, 2006.
- [11] L. Peng, W. Lu, and L. Duan. Power Efficient IP Lookup with Supernode Caching. In *Proc. Globecom*, 2007.
- [12] RIS Raw Data. <http://data.ris.ripe.net>.
- [13] M. A. Ruiz-Sanchez, E. W. Biersack, and W. Dabbous. Survey and taxonomy of IP address lookup algorithms. *IEEE Network*, 15(2):8–23, 2001.
- [14] D. E. Taylor. Survey and taxonomy of packet classification techniques. *ACM Comput. Surv.*, 37(3):238–275, 2005.
- [15] F. Zane, G. J. Narlikar, and A. Basu. CoolCAMs: Power-efficient TCAMs for forwarding engines. In *Proc. INFOCOM*, pages 42–52, 2003.