Architecture-Aware Data Structure Optimization for Green IP Lookup

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

***Abstract*—Power consumption is becoming a limiting factor in next generation network routers. Recent observation shows that IP lookup engines dominate the power consumption of routers. Previous work on reducing power consumption of routers mainly focused on network-, system- and hardware-level optimizations. This paper represents the first attempt on the *data structure* optimization for power-efficient trie-based IP lookup engines. Both non-pipelined and pipelined static random access memory (SRAM) -based architectures are studied. Given the architecture, we formulate the problem by revisiting the time-space trade-off of multi-bit tries. A dynamic programming framework is then proposed to determine the optimal strides for building tree-bitmap tries so that the worst-case power consumption of the IP lookup engine is minimized. Experiments using real-life routing tables demonstrate that careful design of the data structure can reduce the power consumption dramatically. We hope our initial work can motivate the research community to expand their scope beyond the current efforts on either the hardware- or the system- and network- levels for power-efficient Internet infrastructure.**

I. Introduction

The primary function of network routers is to forward packets based on the results of IP lookup, which matches the destination IP address of the packet to the entries in a routing table. As the network traffic grows rapidly, IP lookup becomes a major performance bottleneck for network routers [1], [2]. For example, current backbone link rates have been pushed beyond OC-768 (40 Gbps) rate, which requires a throughput of 125 million packets per second (MPPS) for minimum size (40 bytes) packets. Meanwhile, as routers achieve aggregate throughputs of trillions of bits per second, power consumption by lookup engines becomes an increasingly critical concern in backbone router design [3], [4]. Some recent investigations [5], [6] show that power dissipation has become the major limiting factor for next generation routers and predicts that expensive liquid cooling may be needed in future routers. Recent analysis by researchers from Bell labs [5] reveals that, almost 2/3 of power dissipation inside a core router is due to IP forwarding engines. Most of the recent work focuses on hardware-, system- and network-level optimization for reducing the power consumption of routers [6]–[8]. In this paper we will show that, being aware of the underlying architecture, there remain some opportunities to exploit the data structure optimization for power reduction.

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

Ternary content addressable memories (TCAMs), where a single clock cycle is sufficient to perform an 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 with respect to clock rate, power consumption, and chip density [2]. 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) [3]. Most SRAM-based algorithmic solutions can be implemented as some form of tree traversal, such as trie-based algorithms [9], which can be pipelined to achieve a high throughput of one packet per clock cycle [2]. Hence pipeline architectures have been recently proposed as a promising solution for IP lookup engines in next generation routers [2], [10]. However, they may still suffer from high power consumption, due to the large number of memory accesses [11].

To reduce the number of memory accesses for trie-based algorithms, various multi-bit tries have been proposed [1], [12], [13]. They exhibit trade-off between the memory requirement (space) and the number of memory accesses (time). Either large memory size or a large number of memory accesses leads to high power consumption. It is thus worthwhile to revisit such time-space trade-off from the power/energy point of view. The key contribution of this paper is to study the impact of the data structure design of a multi-bit trie on the power consumption of SRAM-based IP lookup engines. Being aware of the architecture of the engine, a dynamic programming framework is proposed to determine the optimal strides for building a tree-bitmap-coded multi-bit trie so that the worst-case power consumption of the engine is minimized.

The rest of the paper is organized as follows. Section II introduces briefly trie-based algorithms and SRAM-based engines for IP lookup. Section III reviews the efforts on reducing power consumption of routers as well as of IP lookup engines. Section IV studies the data structure optimization to minimize the power consumption of SRAM-based IP lookup engines based on two different architectures. Section V presents the experimental results. Section VI concludes the paper.

II. Background

A. Trie-based IP lookup

The entries in the routing table are specified using prefixes, and IP lookup is to find the longest matching prefix for an

input IP address. The most common data structure used in algorithmic solutions for IP lookup is some form of trie [9]. 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 1 (b) shows the uni-bit trie for the prefix entries in Figure 1 (a). Each trie node contains both the represented prefix and the pointer to the child nodes.

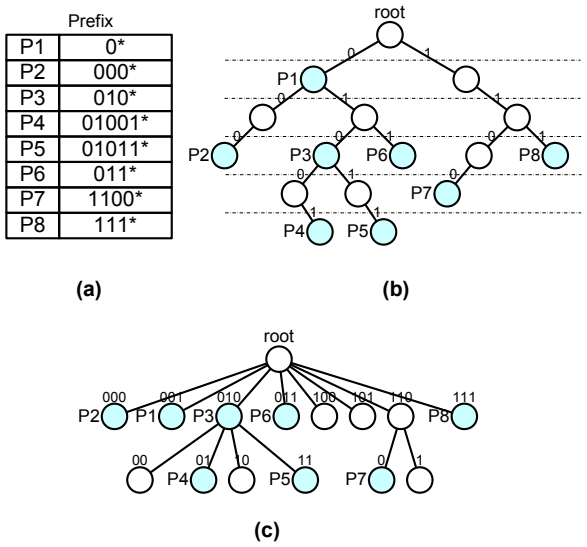

Fig. 1. (a) Prefix entries; (b) Uni-bit trie; (c) Multi-bit trie.

Given a uni-bit trie, LPM is performed by traversing the trie according to the bits in the IP address. The information about the longest prefix matched so far, 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*. Figure 1 (c) shows the multi-bit trie for the prefix entries in Figure 1 (a). The root node uses a stride of 3 while the node containing P3 uses a stride of 2. Multi-bit tries using a larger stride usually result in much larger memory requirement, while some optimization schemes have been proposed for memory compression [1], [12].

The well-known tree bitmap algorithm [1] uses a pair of bit maps for each node in a multi-bit trie. One bit map represents the children that are actually present and the other represents the next hop information associated with the given node. Children of a node are stored in consecutive memory locations, allowing each node to use just a single child pointer. Similarly, another single pointer is used to reference the next hop information associated with a node. This representation allows every node in the multi-bit trie to occupy small memory.

B. Pipelined IP Lookup Engines

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 separate stage, so that a lookup request can be issued every clock cycle [14], [15]. However, such a simple scheme results in unbalanced memory distribution across the pipeline stages. This has been identified as a dominant issue for SRAM-based pipeline architectures [16]. 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 [16]. Although various pipeline architectures [2], [16], [17] have been proposed recently, most of them balance the memory distribution across stages at the cost of lowering the throughput. Our previous work [2] proposes a fine-grained node-to-stage mapping scheme for linear pipeline architectures. It allows the two nodes on the same level to be mapped onto different stages. Balanced memory distribution across pipeline stages is achieved, while a high throughput of one packet per clock cycle is sustained. Figure 2 shows the mapping result for the uni-bit trie in Figure 1 (b) using the fine-grained mapping scheme.

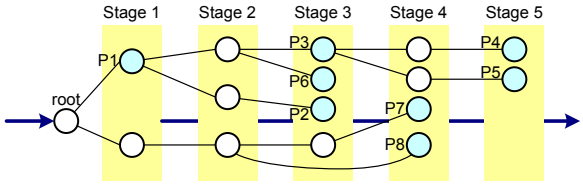

Fig. 2. Mapping the uni-bit trie onto the linear pipeline.

III. Related Work

A. Greening the Routers

Reducing the power consumption of network routers has been a topic of significant interest [4], [6], [8]. Most of the existing work focuses on the system- and network-level optimizations. Chabarek et al. [6] enumerate the power demands of two widely used Cisco routers. The authors further use mixed integer optimization techniques to determine the optimal configuration at each router in their sample network for a given traffic matrix. Nedevschi et al. [8] assume that the underlying hardware in network equipment supports sleeping and dynamic voltage and frequency scaling. The authors propose to shape the traffic into small bursts at edge routers to facilitate sleeping and rate adaptation.

B. Power-Efficient IP Lookup Engines

Power-efficient IP lookup engines have been studied from various aspects. However, to the best of our knowledge, there is little work done for pipelined SRAM-based IP lookup engines. Some TCAM-based solutions [11], [18] propose various

schemes to partition a routing table into several blocks and perform IP lookup on one of the blocks. Similar ideas can be applied for SRAM-based multi-pipeline architectures [19]. Those partitioning-based solutions for power-efficient SRAM-based IP lookup engines do not consider either the underlying data structure or the traffic characteristics, and are orthogonal to the solutions proposed in this paper.

Kaxiras et al. [20] 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 may not be scalable. For large routing tables, the set associativity is still large, resulting in low clock rate and high power consumption.

Traffic rate variation has been exploited in some recent papers for reducing power consumption in multi-core processor based IP lookup engines. In [21] clock gating is used to turn off the clock of unneeded processing engines of multi-core network processors to save dynamic power when there is a low traffic workload. A finer-grained clock gating scheme is proposed in [22] to lower the dynamic power consumption of pipelined IP forwarding engines. Dynamic frequency and voltage scaling are used in [23] and [7], respectively, to reduce the power consumption of the processing engines. However, those schemes still consume large power in the worst case when the traffic rate is consistently high. Some of those schemes require large buffers to store the input packets so that they can determine or predict the traffic rate. But the large packet buffers result in high power consumption. Also, these schemes do not consider the latency for the state transition which can result in packet loss in case of bursty traffic.

IV. Architecture-Aware Data Structure Optimization for Power Efficiency

To the best of our knowledge, little work has been done on data structure optimization for power-efficient SRAM-based IP lookup engines. In this paper we focus on fixed-stride multi-bit tries where all nodes at the same level have the same stride. Fixed-stride multi-bit tries are attractive for hardware implementation due to their ease for route update [1].

We have following notations. Let W denote the maximum prefix length. $W = 32$ for IPv4. Let $S = \{s_0, s_1, \cdots, s_{k-1}\}$ denote the sequence of strides to build a k-level multi-bit trie. Let $|S|$ denote the number of strides in S, i.e. $|S| = k$. $\sum_0^{k-1} s_i = W$. Considering the hardware implementation for tree-bitmap-coded multi-bit tries, we cap the length of strides to be $s_i < B_s, i = 0, 1, \cdots, k-1$, where B_s is a predefined parameter, called the *stride bound*.

A. Problem Formulation for Non-Pipelined Engines

A SRAM-based non-pipelined IP lookup engine stores the entire trie in a single memory. Any IP lookup may need to access the memory for multiple times. Hence the worst-case power consumption of a SRAM-based non-pipelined IP lookup engine can be modeled by Equation (1) where $Power_{memory}$ and $Power_{logic}$ denote the power consumption of the memory and of the logic, respectively.

$$Power = (Power_{memory} + Power_{logic}) \cdot k \tag{1}$$

The logic dissipates much less power than the memories in the memory-intensive architectures [15], [17], [22]. For example, [22] shows that the memory dissipates almost an order of magnitude higher power than the logic in FPGA implementation of a pipelined IP lookup engine. Thus, we do not consider the power consumption of the logic. The optimal stride problem can be formulated as:

$$\min_{k=1,2,\cdots,W} \min_{S(k)} P_m(M(S(k))) \cdot k \tag{2}$$

where $M(S)$ denotes the memory requirement of the multi-bit trie built using S. $P_m(M)$ is the power function of the SRAM of the size M.

B. Problem Formulation for Pipelined Engines

The worst-case power consumption of a SRAM-based pipelined IP lookup engine can be modeled by Equation (3) where H denotes the pipeline depth, i.e. the number of pipeline stages. $Power_{memory}(i)$ and $Power_{logic}(i)$ denote the power consumption of the memory and of the logic in the ith stage, respectively.

$$Power = \sum_{i=1}^{H} [Power_{memory}(i) + Power_{logic}(i)] \tag{3}$$

Similar as for the non-pipelined engine, we omit the power consumption of the logic. Also assuming the memory distribution across the pipeline stage is balanced, the optimal stride problem can be formulated as:

$$\min_S \left\{ [P_m(\frac{M(S)}{H})] \cdot \max(|S|, H) \right\} \tag{4}$$

where $M(S)$ denotes the memory requirement of the multi-bit trie built using S. $P_m(M)$ is the power function of the SRAM of the size M. The number of memory accesses is determined by the $|S|$ and H. When $H < |S|$, multiple clock cycles are needed to access a stage. To achieve high throughput, we let $H \geq |S|$.

Since $|S| = k \leq W$, we can rewrite (4) to be:

$$\min_{k=1,2,\cdots,W} \min_{S(k)} [P_m(\frac{M(S(k))}{H})] \cdot H \tag{5}$$

To solve (2) and (5), we can first fix k and find the optimal $S(k)$ so that the power consumption is minimized for the given k. Then we compare the power consumption for different k to obtain the overall optimal S.

C. Power Function of SRAM

Before we solve the above optimization problem, we need to figure out the power function of the SRAM with respect to its size M: $P_m(M)$. There are some published work on comprehensive power models of SRAM [24]–[26]. But these detailed "white box" models do not show the direct

relationship between the power consumption and the memory size. We use CACTI tool [26] to evaluate both the dynamic and the static power consumption of SRAMs of different sizes and then obtain the function parameters through curve fitting ("black box" modeling).

According to [24], [25], when the word width is constant, both the high-level dynamic and static power consumption of SRAMs can be approximately represented in the form of:

$$P(M) = A \cdot M^B \tag{6}$$

where M is the memory size, and A and B are the parameters whose values are different for dynamic and static power.

We vary the SRAM size from 256 bytes to 8 Mbytes while keeping the word width to be 8 bytes, and obtain their power consumption using CACTI tool [26]. After curve fitting, we obtain that $A_{dynamic} = 2.07 \times 10^{-4}$, $B_{dynamic} = 0.50$, $A_{static} = 1.57 \times 10^{-6}$, $B_{static} = 0.95$. The results from CACTI and curve fitting are both shown in Figure 3.

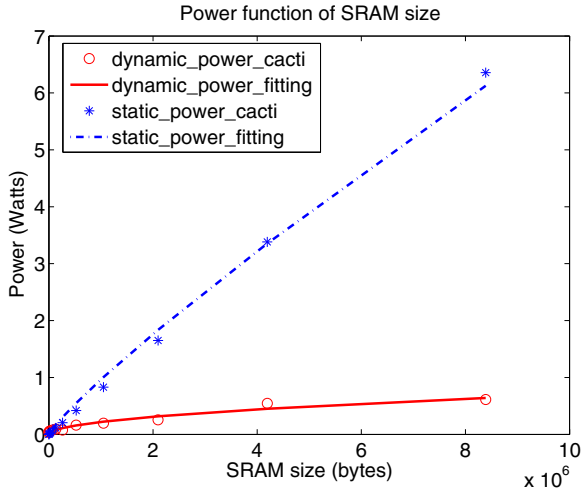

Fig. 3. Power function of SRAM sizes

Hence $P_m(M) = A_{dynamic}M^{B_{dynamic}} + A_{static}M^{B_{static}} \approx 10^{-6}(207M^{0.5} + 1.6M)$. Then (2) and (5) become (7) and (8), respectively:

$$\min_{k=1,2,\cdots,W} \min_{S(k)} (207M(S(k))^{0.5} + 1.6M(S(k))) \cdot k \tag{7}$$

$$\min_{k=1,2,\cdots,W} \min_{S(k)} (207M(S(k))^{0.5}H^{0.5} + 1.6M(S(k))) \tag{8}$$

For a given k, when $M(S(k))$ is minimized, the power consumption is also minimized. Thus the above problems can be reduced to finding the optimal stride so that the memory requirements are minimized.

D. Uniform Stride

In the original tree bitmap paper [1], the authors suggest using the same stride for all the nodes except for the root node. We call such a special fixed-stride multi-bit trie as the multi-bit trie with *uniform stride*. The stride used by the root, i.e. s_0, is called the initial stride. Given k, we can find the optimal S by exhaustive search over different initial strides. In each iteration, $s_i = \left\lceil \frac{W-s_0}{k-1} \right\rceil$, $i = 1, 2, \cdots, k-1$.

E. Dynamic Programming

Srinivasan and Varghese [27] have developed a dynamic programming based solution to minimize the memory requirement of a k-level multi-bit trie. Sahni and Kim [13] made further improvement to reduce the complexity of the algorithms. However, those algorithms focused on the naive implementation of multi-bit tries, without considering the tree bitmap coding technique [1] for compressing the memory requirement of multi-bit tries.

Similar to [27] and [13], we have following notations.

- O denotes the uni-bit trie for the given set of prefixes.
- $nNode(i)$ denotes the number of nodes at the ith level of O.
- $nPrefix(i, j)$ denotes the total number of prefixes contained between the ith and the jth levels of O.
- $T(j, r)$, $r \leq j + 1$, denotes the cost (i.e. the memory requirement) of the best way to cover levels 0 through j of O using exactly r expansion levels.

The dynamic programming recurrence for T is:

$$\begin{aligned} T(j,r) = \min_{m=\max(r-2, j-B_s)}^{j-1} \{ & T(m, r-1) + \\ & nNode(m+1) \cdot 2^{B_s} \cdot 2/32 + \\ & nNode(j+1) + nPrefix(m+1, j) \} \end{aligned} \tag{9}$$

$$T(j, 1) = 2^{j+1} \tag{10}$$

Note that all the strides except the initial stride are capped by the stride bound i.e. B_s. In hardware implementation, the length of the bitmaps is determined by B_s. Algorithm $FixedStride(W, k)$ shown in Figure 4 computes $T(W-1, k)$, the minimum memory requirement to build a k-level tree-bitmap-coded multi-bit trie. The complexity of Algorithm $FixedStride(W, k)$ is $O(k \cdot W \cdot B_s)$. After obtaining $T(W-1, k)$, we can follow the track of the corresponding $M(*, *)$ to find the optimal S in $O(k)$ time.

V. Performance Evaluation

We used 17 real-life backbone routing tables from the Routing Information Service (RIS) [28]. Their characteristics is shown in Table I. Note that the routing tables rrc08 and rrc09 are much smaller than others, since the collection of these two data sets ended on September 2004 and February 2004, respectively [28].

We conducted the experiments for both non-pipelined and pipelined architectures. We evaluated the impacts of different architecture parameters on the power-optimal design of the data structure. The architecture parameters include the stride type, the stride bound, and the pipeline depth. For fixed-stride tree-bitmap-coded multi-bit tries, two stride types are considered. The first uses uniform strides, as described in Section IV-D. The second uses optimal strides whose value is capped by B_s, as discussed in Section IV-E.

Input: O
Output: $T(W-1,k)$, $S(k)$

// Compute $T(W-1,k)$
for $j=0$ to $W-1$ **do**
  $T(j,1)=2^{j+1}$
end for
for $r=2$ to k **do**
  for $j=r-1$ to $W-1$ **do**
    $minCost=MaxValue$
    for $m=\max(r-2,j-B_s)$ to $j-1$ **do**
      $cost=T(m,r-1)+nNode(m+1)\cdot 2_s^B\cdot 2+nNode(j+1)+nPrefix(m+1,j)$
      if $cost<minCost$ **then**
        $T(j,r)=minCost$
        $M(j,r)=m$
      end if
    end for
  end for
end for
// Compute $S(k)=\{s_i\}$, $i=0,1,\cdots,k-1$
$m=W-1$
for $r=k$ to 1 **do**
  $s_{r-1}=m-M(m,r)$
  $m=M(m,r)$
end for

Fig. 4. Algorithm: $FixedStride(W,k)$.

TABLE I
REPRESENTATIVE ROUTING TABLES (SNAPSHOT ON 2009/04/01)

Routing table	# of prefixes	# of prefixes w/ length <16
rrc00	300365	2366 (0.79%)
rrc01	282852	2349 (0.83%)
rrc02	272504	2135 (0.78%)
rrc03	285149	2354 (0.83%)
rrc04	294231	2381 (0.81%)
rrc05	284283	2379 (0.84%)
rrc06	283835	2337 (0.82%)
rrc07	280786	2347 (0.84%)
rrc08	83556	495 (0.59%)
rrc09	132786	991 (0.75%)
rrc10	283573	2347 (0.83%)
rrc11	282761	2350 (0.83%)
rrc12	284469	2350 (0.83%)
rrc13	289849	2355 (0.81%)
rrc14	278750	2302 (0.83%)
rrc15	299211	2372 (0.79%)
rrc16	288218	2356 (0.82%)

A. Results for Non-Pipelined Architecture

First, we set $B_s=4$ and examined the results for the non-pipelined architecture using uniform and optimal strides. The results are shown in Figure 5. In both cases, the power was minimized when $k=5$ and $S=16,4,4,4,4$.

Then we varied the stride bound, i.e. B_s. Figure 6 shows the results using two different stride bounds: $B_s=2$ and $B_s=6$ for the architecture using optimal stride. For $B_s=2$, the power was minimized when $k=9$ and $S=16,2,2,2,2,2,2,2,2$. For $B_s=6$, the minimal power was achieved when $k=4$ and $S=17,5,5,5$.

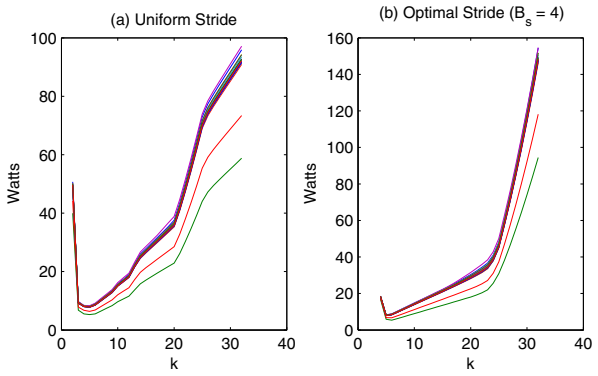

Fig. 5. Power results of the non-pipelined architecture using (a) the uniform stride and (b) the optimal stride.

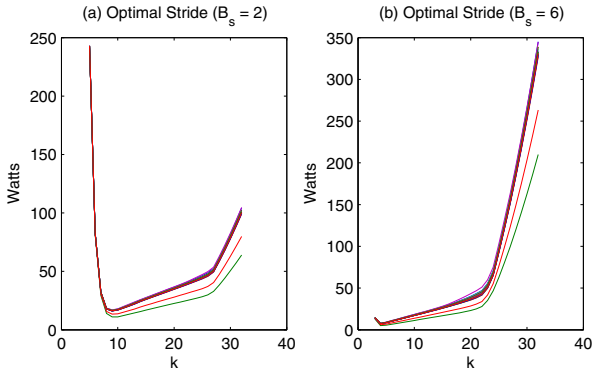

Fig. 6. Power results of the non-pipelined architecture using (a) $B_s=2$ and (b) $B_s=6$.

B. Results for Pipelined Architecture

Figure 7 shows the power consumption of the pipelined architecture using the two stride types. the pipeline depth, i.e. h, was set to be equal to k. The stride bound B_s was 4. Both cases achieved the optimal power performance when $k=6$ and $S=13,4,4,4,4,3$.

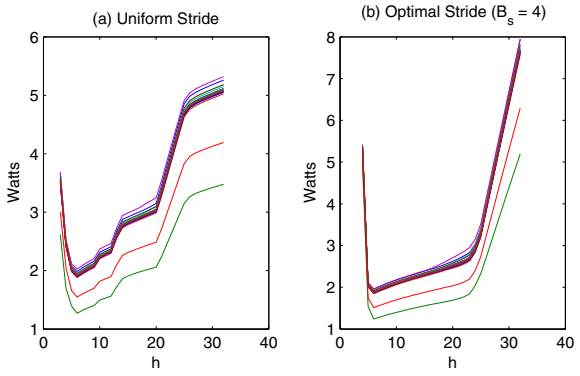

Fig. 7. Power results of the pipelined architecture using (a) the uniform stride and (b) the optimal stride.

Figure 8 shows the results using different stride bounds for the optimal stride. We set $h=k$. For $B_s=2$, the power was minimized when $k=9$ and $S=16,2,2,2,2,2,2,2,2$. For

$B_s = 6$, the minimal power was achieved when $k = 5$ and $S = 16, 4, 4, 4, 4$.

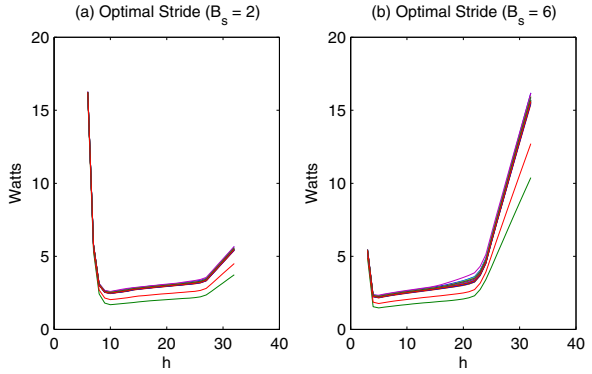

Fig. 8. Power results of the pipelined architecture using (a) $B_s = 2$ and (b) $B_s = 6$.

We also conducted experiments using different pipeline depths. Both cases achieved the minimal power consumption when $k = 6$ and $S = 13, 4, 4, 4, 4, 3$. These are same as the results for $h = k$ (Figure 7(b)). This means the pipeline depth has little impact on determining the optimal strides for pipelined architectures.

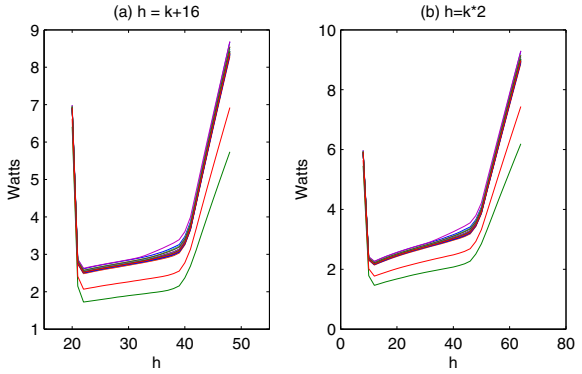

Fig. 9. Power results of the pipelined architecture using (a) $h = k + 16$ and (b) $h = k * 2$.

VI. Conclusion and Future Work

This paper exploited data structure optimization to reduce the power consumption of SRAM-based both non-pipelined and pipelined IP lookup engines. To minimize the worst-case power consumption for a given architecture, a dynamic programming framework was developed to determine the optimal strides for constructing tree bitmap coded multi-bit tries. Simulation using real-life backbone routing tables showed that careful design of the data structure with the awareness of the underlying architecture could achieve dramatic reduction in power consumption. Different architectures could result in different optimal data structures with respect to power efficiency. In the future we plan to study more IP lookup data structures such as the shape shifting trie [12], using the framework developed in this paper. Power-efficient pipeline architectures for IPv6 lookup will also be studied.

References

[1] W. Eatherton, G. Varghese, and Z. Dittia, "Tree bitmap: hardware/software IP lookups with incremental updates," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 97–122, 2004.

[2] 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, pp. 1786–1794.

[3] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 238–275, 2005.

[4] M. Gupta and S. Singh, "Greening of the Internet," in *Proc. SIGCOMM*, 2003, pp. 19–26.

[5] A. M. Lyons, D. T. Neilson, and T. R. Salamon, "Energy efficient strategies for high density telecom applications," *Princeton University, Supelec, Ecole Centrale Paris and Alcatel-Lucent Bell Labs Workshop on Information, Energy and Environment*, June 2008.

[6] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright, "Power awareness in network design and routing," in *Proc. INFOCOM*, 2008, pp. 457–465.

[7] M. Mandviwalla and N.-F. Tzeng, "Energy-efficient scheme for multiprocessor-based router linecards," in *Proc. SAINT*, 2006.

[8] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall, "Reducing network energy consumption via sleeping and rate-adaptation," in *NSDI*, 2008, pp. 323–336.

[9] M. A. Ruiz-Sanchez, E. W. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," *IEEE Network*, vol. 15, no. 2, pp. 8–23, 2001.

[10] L. D. Carli, Y. Pan, A. Kumar, C. Estan, and K. Sankaralingam, "Flexible lookup modules for rapid deployment of new protocols in high-speed routers," in *Proc. SIGCOMM*, 2009.

[11] F. Zane, G. J. Narlikar, and A. Basu, "CoolCAMs: Power-efficient TCAMs for forwarding engines." in *Proc. INFOCOM*, 2003.

[12] H. Song, J. Turner, and J. Lockwood, "Shape shifting trie for faster IP router lookup," in *Proc. ICNP*, 2005, pp. 358–367.

[13] S. Sahni and K. S. Kim, "Efficient construction of multibit tries for IP lookup," *IEEE/ACM Trans. Netw.*, vol. 11, no. 4, pp. 650–662, 2003.

[14] A. Basu and G. Narlikar, "Fast incremental updates for pipelined forwarding engines," in *Proc. INFOCOM*, 2003, pp. 64–74.

[15] J. Hasan and T. N. Vijaykumar, "Dynamic pipelining: making ip-lookup truly scalable," in *Proc. SIGCOMM*, 2005, pp. 205–216.

[16] F. Baboescu, D. M. Tullsen, G. Rosu, and S. Singh, "A tree based router search engine architecture with single port memories," in *Proc. ISCA*, 2005, pp. 123–133.

[17] S. Kumar, M. Becchi, P. Crowley, and J. Turner, "CAMP: fast and efficient IP lookup architecture," in *Proc. ANCS*, 2006, pp. 51–60.

[18] K. Zheng, C. Hu, H. Lu, and B. Liu, "A TCAM-based distributed parallel IP lookup scheme and performance analysis," *IEEE/ACM Trans. Netw.*, vol. 14, no. 4, pp. 863–875, 2006.

[19] W. Jiang and V. K. Prasanna, "Towards green routers: Depth-bounded multi-pipeline architecture for power-efficient IP lookup," in *Proc. IPCCC*, 2008, pp. 185–192.

[20] S. Kaxiras and G. Keramidas, "IPStash: a set-associative memory approach for efficient IP-lookup," in *INFOCOM*, 2005, pp. 992–1001.

[21] Y. Luo, J. Yu, J. Yang, and L. N. Bhuyan, "Conserving network processor power consumption by exploiting traffic variability," *ACM Trans. Archit. Code Optim.*, vol. 4, no. 1, p. 4, 2007.

[22] W. Jiang and V. K. Prasanna, "Reducing dynamic power dissipation in pipelined forwarding engines," in *Proc. ICCD*, 2009.

[23] A. Kennedy, X. Wang, Z. Liu, and B. Liu, "Low power architecture for high speed packet classification," in *Proc. ANCS*, 2008, pp. 131–140.

[24] M. Q. Do, M. Drazdziulis, P. Larsson-Edefors, and L. Bengtsson, "Parameterizable architecture-level SRAM power model using circuit-simulation backend for leakage calibration," in *Proc. ISQED*, 2006, pp. 557–563.

[25] X. Liang, K. Turgay, and D. Brooks, "Architectural power models for SRAM and CAM structures based on hybrid analytical/empirical techniques," in *Proc. ICCAD*, 2007, pp. 824–830.

[26] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, "A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies," in *Proc. ISCA*, 2008, pp. 51–62.

[27] V. Srinivasan and G. Varghese, "Fast address lookups using controlled prefix expansion," *ACM Trans. Comput. Syst.*, vol. 17, pp. 1–40, 1999.

[28] RIS Raw Data, "http://data.ris.ripe.net."