Data Structure Optimization for Power-Efficient IP Lookup Architectures

Weirong Jiang, *Member, IEEE,* and Viktor K. Prasanna, *Fellow, IEEE*

Abstract—Power consumption has become a limiting factor in designing next generation network routers. Recent observation shows that IP lookup engines dominate the power consumption of core routers. Previous work on reducing power consumption of routers mainly focused on network- and system- level optimizations. This paper represents the first thorough study on the data structure optimization for lowering the power consumption in static random access memory (SRAM) -based IP lookup engines. Three different SRAM-based IP lookup architectures are discussed: non-pipelined, simple pipelined, and memory-balanced pipelined architectures. For each architecture, we formulate the problem of power minimization by revisiting the time-space trade-off in multi-bit tries. Two distinct multi-bit trie algorithms are investigated: the expanded trie and the tree bitmap trie, which are widely used in SRAM-based IP lookup solutions. A theoretical framework is proposed to determine the optimal strides for building a multi-bit trie so that the worst-case power consumption of the IP lookup architecture is minimized. Experiments using real-life routing tables including both IPv4 and IPv6 data sets demonstrate that, careful selection of strides in building the multi-bit tries can reduce the power consumption dramatically. We believe our methodology can be applied to other variants of multi-bit tries and can help designing more power-efficient SRAM-based IP lookup architectures.

Index Terms—IP lookup, data structure, power-efficient, pipeline, SRAM.

1 Introduction

The primary function of network routers is to forward packets based on the results of *IP lookup*, which retrieves the next-hop information by matching the destination IP address of a packet to the entries in a routing table. As the network traffic keeps growing rapidly, IP lookup has become a major performance bottleneck for network routers [1], [2]. For example, current backbone link rates have been pushed towards 100 Gbps rate [3], which requires a throughput of 312.5 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 core router design [4], [5]. Some recent investigations [6], [7] show that power dissipation has become the major limiting factor and predicts that expensive liquid cooling may be needed in next generation core routers. Several recent work proposes various system- and network-level optimizations for reducing the power consumption of routers [7], [8]. But they remain insufficient to address the challenge of high power consumption for core routers in the worst case (i.e. full-load traffic).

Recent analysis by researchers from Bell labs [6] reveals that, almost two thirds of power dissipation inside a core router is due to IP lookup engines. To meet the high throughput requirement in backbone, it becomes a must to perform IP lookup in hardware. Current hardware-based IP lookup solutions can be divided mainly into two categories: Ternary Content Addressable Memory (TCAM)-based and Static Random Access Memory (SRAM)-based. TCAM-based solutions, where a single clock cycle is sufficient to perform an IP lookup, are widely used in today's edge 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, and chip density [2]. It has been estimated that the power consumption per bit of TCAMs is on the order of 3 micro-Watts, which is 150 times more than for SRAM [4]. As a result, today's core routers such as Juniper's T1600 [9] and Cisco's CRS-3 [10] routers implement trie-based IP lookup algorithms in SRAM-based hardware architectures.

Most SRAM-based algorithmic solutions are based on *trie* [11], whose search process can be pipelined to achieve a high throughput of one packet per clock cycle [2]. SRAM-based pipeline architectures have been known as an attractive solution for IP lookup engines in next generation routers [2], [12]. However, SRAM-based IP lookup engines still suffer from high power consumption, due to the large number of accesses on large memory [14]. Hence the main focus of this paper is on designing power-efficient SRAM-based IP lookup engines.

We revisit the classical IP lookup data structure: the trie. Various multi-bit tries have been proposed to reduce the number of memory accesses for trie-

- *W. Jiang is with the Juniper Networks Inc., Sunnyvale, CA, 94089, USA, e-mail: weirongj@acm.org.*
- *V. K. Prasanna is with the Ming Hsieh Department of Electrical Engineering, University of Southern California, Los Angeles, CA, 90007, USA, e-mail: prasanna@usc.edu.*

Supported by the U.S. NSF under grant No. CCF-1116781.

Digital Object Indentifier 10.1109/TC.2012.199

based algorithms [1], [17], [18]. They exhibit trade-off between the memory size (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 main contributions of this paper include a thorough study on the impact of the data structure tuning of a multi-bit trie on the power consumption of SRAM-based IP lookup architectures. The tuning knob is the strides used to build a multi-bit trie. We study two existing multi-bit trie algorithms including the expanded trie [19] and the tree bitmap trie [1], which are among the most well-known multi-bit trie algorithms that have been used in today's core routers. Three different IP lookup architectures are discussed: non-pipelined, simple pipelined, and memory-balanced pipelined architectures. A theoretical framework is proposed to determine the optimal strides for building a multi-bit trie so that the worst-case power consumption of the architecture is minimized. Both IPv4 and IPv6 backbone routing tables are evaluated in our experiments to verify the effectiveness of our solution.

The rest of the paper is organized as follows. Section 2 gives a overview of trie-based IP lookup algorithms and introduces SRAM-based IP lookup architectures. Section 3 defines and formulates the problems of power minimization. Section 4 details our solutions. Section 5 presents the experimental results. Section 6 reviews the recent efforts on reducing power consumption of routers as well as of IP lookup engines. Section 7 concludes the paper.

TABLE 1
Example Prefix Set

P1	0*
P2	1*
P3	10*
P4	111*
P5	1000*
P6	11001*
P7	100000*
P8	1000000*

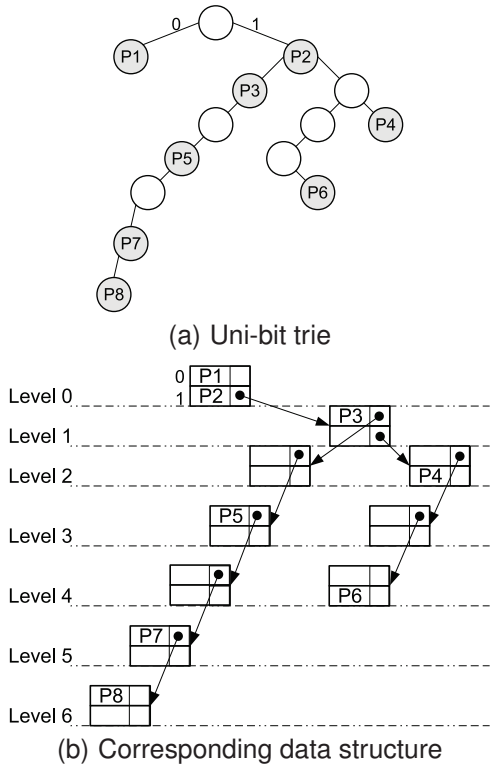


Fig. 1. A uni-bit trie and its data structure

2 BACKGROUND

2.1 Trie-based IP lookup

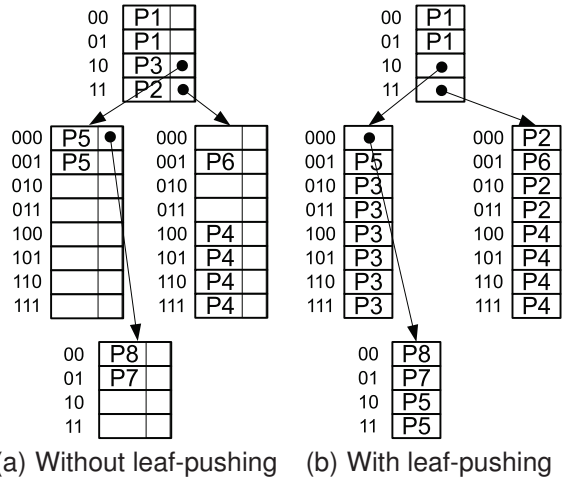
The entries in a routing table are specified using prefixes. IP lookup is to find the longest matching prefix for an input IP address. The most common data structure used in algorithmic solutions for IP lookup is some form of trie [11]. A basic binary trie is a binary tree, where a prefix is represented 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 shows the uni-bit trie for the prefix entries in Table 1. Figure 1(a) gives a classical representation of the uni-bit trie, while Figure 1(b) illustrates the actual data structure of a uni-bit trie. Each trie node contains both the pointer to the child nodes and the pointer to the next-hop information associated with the represented prefix. By using the *leaf-pushing* [19] technique, each node needs only one field: either the pointer to the next-hop address or the pointer to the child nodes.

Given a uni-bit trie, longest prefix matching (LPM) is performed by traversing the trie according to the bits in the IP address. When a leaf is reached, the longest matching prefix along the traversed path is returned. The time to look up a uni-bit trie is up to the maximum prefix length. In the worst case, it takes 32 and 128 memory accesses to find the longest matching prefix for IPv4 (32-bit) and IPv6 (128-bit), respectively.

The search speed can be improved by using multiple bits in one scan when traversing the trie. This results in a *multi-bit trie*. The number of bits scanned at a time is called the *Stride*. There are two versions of multi-bit tries: fixed-stride and variable-stride tries. The nodes at the same level have the same stride in a *fixed-stride* trie, while they may have different strides in a *variable-stride* trie. Fixed-stride tries are more desirable for hardware implementation due to their simplicity and ease of route update [1], [19]. Hence this paper considers only the fixed-stride tries.

A naive implementation of a multi-bit trie is the Expanded trie [19]. Figure 2 shows the fixed-stride

expanded trie for the prefix entries in Table 1 with the strides of 2,3,2. That is, the first level of the trie uses two bits, the second uses three bits, and the third uses two bits. Figures 2(a) and 2(b) show the expanded tries without and with leaf-pushing, respectively.



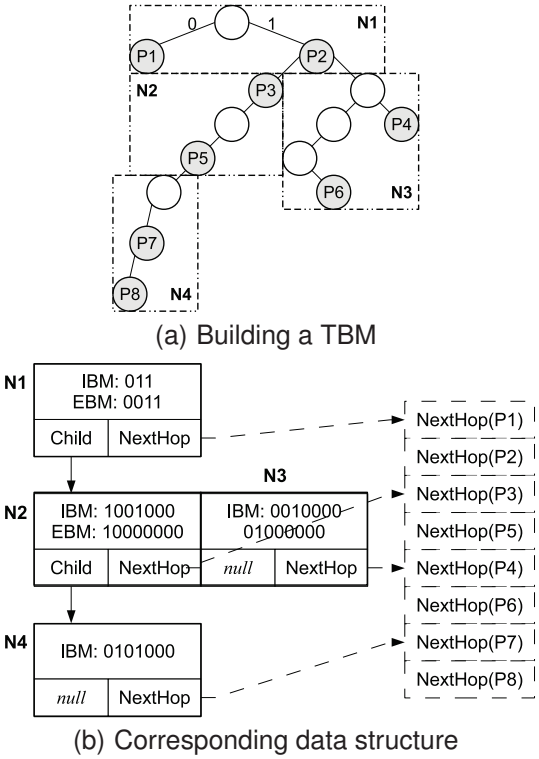
(a) Without leaf-pushing (b) With leaf-pushing

Fig. 2. Expanded trie (fixed-stride)

Expanded tries are not memory-efficient. Various optimization schemes have been proposed for memory reduction [1], [17]. The most well-known (and successful) one is the tree bitmap (TBM) algorithm [1] which reduces memory requirement dramatically by employing a clever encoding of a fixed-stride multi-bit trie. The TBM algorithm uses a pair of bitmaps for each node in a TBM trie. One bitmap (named internal bitmap, denoted as IBM) represents the next-hop information associated with the internally stored prefixes inside the given multi-bit trie node. The other bitmap (named external bitmap, denoted as EBM) represents the children that are actually present. For a TBM node using a stride of s, its IBM has $2^s - 1$ bits and its EBM has 2^s bits. If the TBM node is an *end node* (whose children are all leaf nodes), its EBM can be eliminated and its IBM is expanded to $2^{(s+1)} - 1$ bits. Children of a node are stored in contiguous memory locations, which allows each node to use just a single child pointer. The memory address of each child node can be calculated as an offset from the single pointer. Similarly, another single pointer is used to reference the next-hop information associated with the prefixes inside a node. Figure 3 shows the TBM trie corresponding to the prefix entries in Table 1 with the strides of 2,3,2. Figure 3(a) shows how a TBM is built based on the reference uni-bit trie. There are four TBM nodes, denoted as N1, N2, N3 and N4, where N3 and N4 are end nodes. Figure 3(b) shows the corresponding data structure of the TBM trie.

2.2 SRAM-based IP Lookup Architectures

Trie-based algorithmic IP lookup solutions can be implemented in SRAM-based hardware architectures



(a) Building a TBM

(b) Corresponding data structure

Fig. 3. A TBM and the corresponding data structure

to achieve high performance. A traditional method is to store the entire trie into a single SRAM chip. It needs to access the single memory multiple times for looking up an IP address. Thus the worst-case throughput for searching a K-level trie is $1/K$ packets per clock cycle, given that each SRAM access takes one clock cycle.

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 [20], [21]. However, such a simple scheme results in unbalanced memory distribution across the pipeline stages. This has been identified as a major issue for SRAM-based pipeline architectures [22], [23], [24]. In an unbalanced pipeline, more time is needed to access the larger local memory. This leads to a reduction in the global clock rate.

Various memory-balanced pipeline architectures have been proposed recently [2], [22], [25]. But most of them balance the memory distribution across stages at the cost of lowering the throughput, due to their non-linear architectures. 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. This is enabled by storing in each node the distance to the stage where its child nodes reside. Balanced memory distribution across pipeline stages is achieved, while a throughput of one packet per clock cycle is sustained.

Figure 4 depicts the examples of the above three SRAM-based IP lookup architectures. Each of them

has its pros and cons. The non-pipelined architecture is the easiest to implement and to support fast route update, but with the lowest throughput. The memory-balanced linear pipelined architecture achieves the highest throughput with the highest complexity that leads to the difficulty in handling route updates. This paper does not aim to make any comparison between the three architectures. We want to see how the data structure optimization will impact the power efficiency over different architectures.

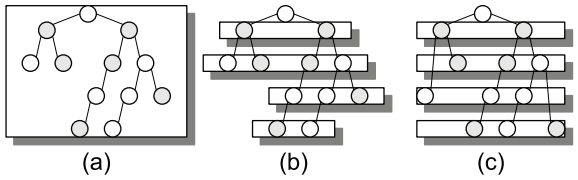


Fig. 4. (a) Non-pipelined (NP), (b) Simple pipelined (SP), and (c) Memory-balanced pipelined (MBP) architectures

3 Problem Formulation

3.1 Notations

We have following notations:

- K: The number of trie levels.
- s: A stride.
- S_K: The strides to build a K-level trie. $S_K = \{s_0, s_1, \ldots, s_{K-1}\}$, where s_i is the stride for building the i-th level of the trie, $i = 0, 1, \ldots, K-1$.
- H: The number of stages in a pipeline.
- p_i: The power dissipation of the i-th stage in the pipeline, $i = 1, 2, \ldots, H$.
- N_w: The number of memory words.
- $N_w[i]$: The number of words on the i-th level of the trie, $i = 0, 1, \ldots, K-1$.
- W_w: The width of a memory word, in terms of the number of bits.
- $W_w[i]$: The word width for the nodes on the i-th level of the trie, $i = 0, 1, \ldots, K-1$.
- $P_m(N_w, W_w)$: The power dissipation of the memory as a function of N_w and W_w.
- $M(S_K)$: The memory size of the trie constructed using the strides S_K.
- c: A constant number.

When building a K-level trie, N_w and W_w may be determined by the strides S_K. Hence they can be represented as $N_w(S_K)$ and $W_w(S_K)$, respectively. But for clearness, we still use the notation of N_w and W_w which implicitly mean $N_w(S_K)$ and $W_w(S_K)$, respectively, in the rest of the paper.

3.2 Power Modeling

3.2.1 Assumption

The power consumption of a SRAM-based architecture includes both the power dissipation of the memory and of the logic. Several previous work [15], [16], [25] has shown that the logic dissipates much less power than the memory in a memory-intensive IP lookup architecture. Hence the main focus of this paper is on reducing the power consumption caused by memory accesses. The power dissipation due to the logic will be ignored in the rest of the paper.

3.2.2 Power Consumption Metrics

We consider two metrics to evaluate the power consumption of an IP lookup architecture:

1) Power dissipation of the hardware architecture. Denoted as P_1.
2) Maximum power consumed by an IP packet going through the architecture. Denoted as P_2.

The relationship between P_1 and P_2 is different for different architectures:

In a non-pipelined architecture containing a K-level trie, an IP lookup may need access the architecture for K times. Thus $P_2 = K \cdot P_1$. As $K \geq 1$, $P_1 \leq P_2$.

In a simple pipelined architecture, $H = K$. Both P_1 and P_2 are equal to the sum of power dissipation of all stages. That is, $P_1 = P_2 = \sum_{i=1}^{H} p_i$.

In a memory-balanced pipelined architecture, $H \geq K$. As the memory is balanced across the stages, the power dissipation of any stage is the same, i.e. $p_i = p$, $i = 1, 2, \ldots, H$. $P_1 = p \cdot H$ and $P_2 = p \cdot K$. $P_1 \geq P_2$.

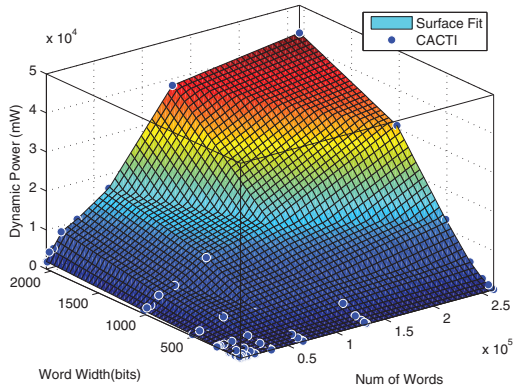
3.2.3 Power Function of SRAM

We need to figure out the power function of the SRAM with respect to its parameters. The only memory-related parameters available from the trie data structure are the number of memory words (N_w) and the word width (W_w). There are some published work on comprehensive power models of SRAM [26], [27], [28]. But these detailed analytic models do not give the explicit relationship between the power consumption and (N_w, W_w). An earlier version of our work [29] considered N_w as the only variable and attempted to obtain the power function of SRAM with respect to the memory size (which is $N_w \cdot W_w$) using the fixed word width (W_w = 64 bits). CACTI tool [28] (version 5.3) was used to evaluate both the dynamic and the leakage power consumption of SRAMs of different sizes. Then the function parameters were obtained through curve fitting ("black box" modeling). It was revealed that, when the word width was constant, the dynamic and the leakage power of SRAM were sublinear and linear to the memory size, respectively.

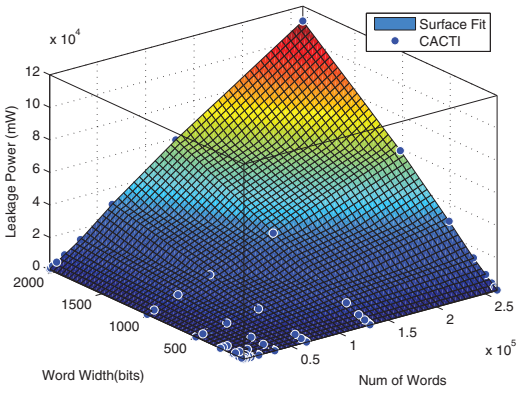
We employ the same methodology as [29] to profile the power function of SRAM with respect to (N_w, W_w). As there are two instead of one variables, we have to use surface fitting instead of curve fitting. As shown in Figure 5(b), the leakage power is linear to both N_w and W_w. However, while Figure 5(a) shows that the dynamic power of SRAM is not linear to either N_w or W_w, it is difficult to find a good fitting function. Hence, we leave the power function

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

of SRAM as a black box, denoted as $P_m(N_w, W_w)$. We link the CACTI module [28] into our algorithms. The CACTI module receives the values of N_w and W_w, and outputs the power dissipation of the SRAM. Other CACTI parameters are fixed, as listed in Table 2.



(a) Dynamic power (mW)



(b) Leakage power (mW)

Fig. 5. Power of SRAM as a function of (N_w, W_w)

TABLE 2
Default CACTI parameters

Name	Value
Number of Banks	1
Number of Read/Write Ports	1
Number of Read Ports	0
Number of Write Ports	0
Technology Node	65 nm
Temperature	360 K
Transistor type	ITRS-HP
Interconnect projection type	Conservative
Type of wire	Semi-global

Note that $P_m(.)$ is not a continuous function to N_w. This is because the number of words must be power of 2 in a physical memory whose parameters are the inputs to the SRAM power function, while N_w mainly refers to the number of words consumed by the data structure of the trie. For example, in case of fixed W_w, $P_m(513, W_w) = P_m(1023, W_w)$, because either $N_w = 513$ or $N_w = 1023$ words need to be stored in a 1024-word memory.

3.3 Problem of Power Minimization

We aim to minimize the power consumption of SRAM-based IP lookup architectures by choosing the optimal strides in building a fixed-stride multi-bit trie. Given a set of prefixes and a trie algorithm, we need to determine the number of strides (K) and the value of each stride (S_K). The problem can be formulated as (1) and (2):

$$\min_K \min_{S_K} P_1 \tag{1}$$

$$\min_K \min_{S_K} P_2 \tag{2}$$

The problems can be detailed for the three different architectures including: the non-pipelined (*NP*), the simple pipelined (*SP*), and the memory-balanced pipelined (*MBP*) architectures.

3.3.1 Problem for NP Architecture

In a non-pipelined (NP) architecture, P_1=$P_m(N_w, W_w)$ and P_2=$K \cdot P_1$. Then (1) and (2) can be rewritten as:

$$\min_K \min_{S_K} P_1 = \min_K \min_{S_K} P_m(N_w, W_w) \tag{3}$$

$$\begin{aligned} \min_K \min_{S_K} P_2 &= \min_K \min_{S_K} K \cdot P_m(N_w, W_w) \\ &= \min_K K \cdot \min_{S_K} P_m(N_w, W_w) \end{aligned} \tag{4}$$

The basic problem to be solved, which is common to both (3) and (4), is

$$\min_{S_K} P_m(N_w, W_w) \tag{5}$$

which is to identify the optimal strides in building a K-level trie so that the power dissipation of the resulting memory is minimized. Once this basic problem is solved, we can iterate all possible values of K to find out the optimal K.

Note that in a NP architecture, the nodes on all levels of a trie are stored in a single memory. Thus the word width of the memory has to be identical and should be determined by the largest word width across all levels of the trie. In other words,

$$W_w = \max_{i=0}^{K-1} W_w[i]. \tag{6}$$

3.3.2 Problem for SP Architecture

In a simple pipelined (SP) architecture, $P_1 = P_2 = \sum_{i=1}^{K} p_i$ where $p_i = P_m(N_w[i], W_w[i])$, $i = 0, 1, \ldots, K-1$. Then (1) and (2) can be rewritten as (7):

$$\begin{aligned} &\min_K \min_{S_K} P_1 = \min_K \min_{S_K} P_2 = \\ &\min_K \min_{S_K} \sum_{i=1}^{K} P_m(N_w[i], W_w[i]) \end{aligned} \tag{7}$$

The basic problem to be solved is

$$\min_{S_K} \sum_{i=1}^{K} P_m(N_w[i], W_w[i]) \tag{8}$$

which is to identify the optimal strides for building a K-level trie so that after each level of the trie is mapped to a separate memory block, the sum of the power dissipation of the K memory blocks is minimized. Once this basic problem is solved, we can iterate all possible values of K to get the optimal K.

3.3.3 Problem for MBP Architecture

In a memory-balanced pipelined (MBP) architecture, $P_1 = p \cdot H$ and $P_2 = p \cdot K$. Since the memory distribution across the pipeline stages is balanced, $p = P_m(\frac{N_w}{H}, W_w)$. As H needs to be larger than K, we have $H = K + \Delta H$, where ΔH denotes the extra number of stages added upon K. Then (1) and (2) can be rewritten as

$$\begin{aligned}\min_K \min_{S_K} P_1 &= \min_K \min_{S_K} P_m(\frac{N_w}{H}, W_w) \cdot (K + \Delta H) \\ &= \min_K (K + \Delta H) \cdot \min_{S_K} P_m(\frac{N_w}{H}, W_w) \quad (9)\end{aligned}$$

$$\begin{aligned}\min_K \min_{S_K} P_2 &= \min_K \min_{S_K} P_m(\frac{N_w}{H}, W_w) \cdot K \\ &= \min_K K \cdot \min_{S_K} P_m(\frac{N_w}{H}, W_w) \quad (10)\end{aligned}$$

The basic problem to be solved, which is common to both (9) and (10), is

$$\min_{S_K} P_m(\frac{N_w}{H}, W_w) \quad (11)$$

which is to identify the optimal strides for building a K-level trie so that after the trie nodes are uniformly distributed across H stages, the power dissipation of each stage is minimized. Once this basic problem is solved, we can iterate all possible values of K to obtain the optimal K.

Note that in a memory-balanced pipelined architecture, the nodes on different levels of a trie can be stored in a same memory. The only exception is the root node. Thus the word width in all but the first stages has to be identical to be

$$W_w = \max_{i=1}^{K-1} W_w[i]. \quad (12)$$

4 Solution Techniques

According to the discussion in Section 3.3, there are three basic problems to be solved, which are listed as (5), (8) and (11). Each problem corresponds to one of the three architectures. In this section, we solve the three problems for two variants of multi-bit tries: the expanded trie and the tree bitmap (TBM) trie.

Lemma 1: When the word width (W_w) is constant, the power dissipation of a single SRAM is minimized if and only if its memory size (M) is minimized.

Proof: $M = N_w \cdot W_w$. When W_w is constant i.e. $W_w = c$, $\min M \Leftrightarrow \min N_w$. On the other hand, the power dissipation of the SRAM i.e. $P_m(N_w, W_w)$, becomes $P_m(N_w, c)$. According to the SRAM power function, $\min P_m(N_w, c) \Leftrightarrow \min N_w$.

Thus, $\min P_m(N_w, c) \Leftrightarrow \min M$. □

4.1 Power Minimization for Expanded Trie

In an expanded trie as shown in Figure 2, strides only affects N_w, while W_w is independent with the strides. More precisely, an expanded trie node using a stride of s contains 2^s words. On the other hand,

$$W_w = W_w[i] = c, \quad i = 0, 1, \ldots, K-1, \quad (13)$$

where c is a constant.

4.1.1 Solving the Problem for NP Architecture

Based on (13) and Lemma 1, the problem (5) can be reduced to

$$\min_{S_K} M(S_K) \quad (14)$$

which is to find the optimal strides for building a minimum-memory K-level expanded trie. This problem has been well studied by Srinivasan and Varghese [19], and Sahni and Kim [18]. Srinivasan and Varghese developed a dynamic programming solution to minimize the memory requirement of a K-level expanded trie. Sahni and Kim made further improvement to reduce the complexity of the algorithms. Since our solutions are based on their work, we briefly reproduce the idea of the dynamic programming solution proposed in [19]. First, we have the following notations in addition to those defined in Section 3.1:

- O: The uni-bit trie for the given set of prefixes.
- E: The K-level expanded trie for the same set of prefixes.
- L: The maximum prefix length. Note that the number of levels of O equals L.
- $nn(i)$: The number of nodes on Level i of O, $i = 0, 1, \ldots, L-1$.

Each level of E is called an *expansion level*, as it covers multiple levels of O. Consider E uses the strides of $S_K = \{s_0, s_1, \ldots, s_{K-1}\}$. Level 0 of E covers Levels 0,...,$s_0 - 1$ of O. Level j of E, $j = 1, 2, \ldots, K-1$, covers Levels $\sum_{q=0}^{j-1} s_q$,...,$\sum_{q=0}^{j} s_q - 1$ of O.

Let $T(j, r)$ be the optimal cost (i.e. memory requirement) to cover Levels 0 through j of O using r expansion levels. Then $T(L-1, K)$ is the cost of the best K-level expanded trie for the given prefix set. The following dynamic programming recurrence is obtained in [19]:

$$\begin{aligned}T(j, r) &= \min_{m \in [r-2, j-1]} \{T(m, r-1) + \\ &\qquad nn(m+1) * expCost(j-m)\} \quad (15) \\ T(j, 1) &= expCost(j+1)\end{aligned}$$

where $expCost(s) = 2^s$ is the expansion cost (i.e. memory requirement in terms of the number of words) of using the stride of value s.

The complexity of the dynamic programming algorithm is $O(KL^2)$. Since the maximum value of K can be L, the complexity of the algorithms to solve the problems (3) and (4) for the expanded trie is $\sum_{K=1}^{L} O(KL^2) = O(L^4)$.

4.1.2 Solving the Problem for SP Architecture

As the word width is constant in an expanded trie, i.e. $W_w = c$, the problem (8) can be reduced to

$$\min_{S_K} \sum_{i=1}^{K} P_m(N_w[i]) \tag{16}$$

Note that this problem is not equal to the problem of

$$\min_{S_K} \sum_{i=1}^{K} N_w[i] = \min_{S_K} N_w \Leftrightarrow \min_{S_K} M(S_K).$$

This is because the power dissipation of SRAM is not linear to N_w, as shown in Figure 5(a).

To solve (16), we use the similar dynamic programming recurrence as (15) but with a different expansion cost function $expCost(s)$ for stride s. By integrating the power function of SRAM, the new expansion cost function is

$$expCost(s) = P_m(2^s, c)$$

The complexity of the dynamic programming algorithm is $O(KL^2)$. Thus the complexity of the algorithm to solve the problem (7) for the expanded trie is $\sum_{K=1}^{L} O(KL^2) = O(L^4)$.

4.1.3 Solving the Problem for MBP Architecture

As W_w is constant in an expanded trie, the problem (11) can be reduced to (17) which is equal to (14):

$$\min_{S_K} P_m(\frac{N_w}{H}) \Leftrightarrow \min_{S_K} N_w \Leftrightarrow \min_{S_K} M(S_K) \tag{17}$$

Thus the solution for MBP architecture is same as that for NP architecture. But this does not mean that the results (i.e. the optimal K and S_K) are the same for the two architectures. The complexity of the algorithms to solve (9) and (10) for the expanded trie is $O(L^4)$.

4.2 Power Minimization for TBM Trie

As far as we know, there is no previous work on identifying the optimal strides in building a TBM trie to achieve either memory or power minimization. In a TBM trie (shown in Figure 3), each node is stored as a word. Each word contains the bitmaps whose size depends on the stride value s. There are two kinds of nodes in a TBM trie: internal nodes and end nodes. Both kinds of nodes have the same word width which can be represented as a function of s:

$$W_w(s) = 2^{(s+1)} + c, \tag{18}$$

where c is a constant.

4.2.1 Solving the Problem for NP Architecture

As discussed in Section 3.3.1, all nodes in a non-pipelined (NP) architecture have to use the same word width whose value is determined by the largest node in the TBM trie. Based on (18), we can rewrite (6) as:

$$\begin{aligned} W_w &= \max_{i=0}^{K-1} W_w[i] = \max_{i=0}^{K-1} 2^{(s_i+1)} + c \\ &= 2^{(\max_{i=0}^{K-1} s_i+1)} + c. \end{aligned} \tag{19}$$

So the word width is determined by the largest stride. Since the word width depends on the strides, Lemma 1 is no longer applicable directly to the TBM trie. Moreover, as the largest stride is unknown during the course of building the trie, it is difficult to evaluate the expansion cost if we use the similar dynamic programming recurrence as in previous sections. To solve the problem, we add a *stride bound* (B) as the upper limit of any stride when building a K-level TBM trie. In other words, we build a K-level TBM trie whose strides are capped by B, i.e. $\max_{i=0}^{K-1} s_i \leq B$. Let $S_{K,B}$ denote the bounded strides: $\{s_0, s_1, \ldots, s_{K-1}\}$ where $s_i \leq B$, $i = 0, 1, \ldots, K-1$. Then we transform the problem (5) to be

$$\min_{B} \min_{S_{K,B}} P_m(N_w, W_w(B)) \tag{20}$$

Now the basic problem to be solved becomes

$$\min_{S_{K,B}} P_m(N_w, W_w(B)) \tag{21}$$

Once this basic problem is solved, we can iterate all possible values of B to get the optimal B and the corresponding $S_{K,B}$. Given a B, W_w is fixed to be $2^{(B+1)} + c$. Thus similar to the case for the expanded trie (Section 4.1.1), we can reduce the problem (21) to be the problem of $\min_{S_{K,B}} M(S_{K,B})$. Then we can solve it by using the dynamic programming recurrence similar to (15). But as the strides are capped by B, we need to revise it to be

$$\begin{aligned} T(j,r) &= \min_{m \in [max(r-2, j-B), j-1]} \{T(m, r-1) + \\ &\quad nn(m+1) * expCost(B)\} \\ T(j,1) &= expCost(B), j < B \\ T(j,1) &= +\infty, j \geq B \end{aligned} \tag{22}$$

where $expCost(s) = W_w(s)$ is the expansion cost (i.e. memory requirement in terms of the number of bits) of using the stride s. We employ $T(j,1) = +\infty, j \geq B$ to cap s_0. The complexity of the dynamic programming algorithm is $O(KL^2)$. As the maximum value of B can be L, the complexity of the algorithm to solve the problem (20) is $\sum_{B=1}^{L} O(KL^2) = O(KL^3)$. The complexity of the algorithms to solve the problems (3) and (4) for the TBM trie is $\sum_{K=1}^{L} O(KL^3) = O(L^5)$.

Note that, though $\min_{S_{K,B}} P_m(N_w, W_w(B)) \Leftrightarrow \min_{S_{K,B}} M(S_{K,B})$, the problem (20) is not equal to $\min_B \min_{S_{K,B}} M(S_{K,B})$. This is because $M = N_w \cdot W_w$ but $P_m(N_w, W_w)$ is not linear to M. The B that results in the minimum memory does not necessarily lead to the minimum power consumption.

4.2.2 Solving the Problem for SP Architecture

In a simple pipelined (SP) architecture, the trie nodes on the same level are mapped to the same stage. Since a TBM trie is a fixed-stride trie, the trie nodes on the same level use the same stride. Thus the words stored in a same stage have the same word width. On the

other hand, different stages can use different strides. Thus there is no need for the stride bound.

For the case of expanded trie (Section 4.1.2), N_w is the only variable. Now for the TBM trie, we need to consider both N_w and W_w. We propose the following dynamic programming recurrence to solve the problem (8) for the TBM trie:

$$\begin{aligned} T(j,r) &= \min_{m\in[r-2,j-1]}\{T(m,r-1)+ \\ &\quad P_m(nn(m+1), expCost(j-m))\} \quad (23) \\ T(j,1) &= P_m(nn(0), expCost(j+1)) \end{aligned}$$

The complexity of the dynamic programming algorithm is $O(KL^2)$. Thus the complexity of the algorithm to solve the problem (7) for the TBM trie is $\sum_{K=1}^{L} O(KL^2) = O(L^4)$.

4.2.3 Solving the Problem for MBP Architecture

As discussed in Section 3.3.3, the trie nodes on different levels may be mapped to the same stage in a memory-balanced pipelined (MBP) architecture. The memory words in all stages except the first stage must use the same word width. Similar to the discussion for a non-pipelined (NP) architecture (Section 4.2.1), the word width in those stages is determined by the largest stride among the strides (excluding s_0) of the TBM trie. But the largest stride is unknown until the trie is constructed. Like the case for the NP architecture, we can solve the problem by adding the stride bound (B). But we do not need to cap the first stride (s_0) in the MBP architecture. Let $S'_{K,B}$ denote the bounded strides: $\{s_0, s_1, \ldots, s_{K-1}\}$ where $s_i \leq B$, $i = 1, 2, \ldots, K-1$. Then the problem (11) is transformed to be

$$\min_{B} \min_{S'_{K,B}} P_m(\frac{N_w}{H}, W_w(B)) \quad (24)$$

The basic problem to be solved becomes

$$\min_{S'_{K,B}} P_m(\frac{N_w}{H}, W_w(B)) \quad (25)$$

Given a B, W_w is fixed to be $2^{(B+1)} + c$. Thus similar to the case for the expanded trie (Section 4.1.3), we can reduce the problem (25) to be the problem of $\min_{S'_{K,B}} M(S'_{K,B})$. The solution is very similar to that for the NP architecture (Section 4.2.1), with the only difference is that the stride bound for s_0 is removed here. So we have:

$$\begin{aligned} T(j,r) &= \min_{m\in[max(r-2,j-B),j-1]}\{T(m,r-1)+ \\ &\quad nn(m+1) * expCost(B)\} \\ T(j,1) &= expCost(B) \end{aligned} \quad (26)$$

Same as the analysis in Section 4.2.1, the complexity of the algorithms to solve the problems (9) and (10) for the TBM trie is $O(L^5)$.

5 Experimental Results

We conduct the experiments for the three SRAM-based IP lookup architectures. For each architecture, both expanded trie and tree bitmap (TBM) trie are evaluated. The stride selection includes the number of strides (K) and the values of strides (S_K). We'd like to see which K and S_K lead to the best performance for each architecture with each type of trie. Three performance metrics are considered:

- Memory requirement, denoted as Mem;
- Power dissipation of the architecture, i.e. P_1 as defined in Section 3.2.2;
- Worst-case power consumption by an IP lookup, i.e. P_2 as defined in Section 3.2.2.

5.1 Data Set

We use 17 real-life backbone routing tables from the Routing Information Service (RIS) [30]. Most of the routing tables contain both IPv4 and IPv6 prefixes. We divide each routing table into a IPv4 prefix set and a IPv6 prefix set. Table 3 lists their characteristics including the numbers of unique IPv4 and IPv6 prefixes. The empty prefix sets are not used. Hence we have 17 IPv4 prefix sets and 14 IPv6 prefix sets. Note that the routing tables rrc02, rrc08 and rrc09 are much smaller than others, since the collection of these three data sets ended on October 2008, September 2004 and February 2004, respectively [30].

TABLE 3
Representative Routing Tables

Routing table	Site	Date & Time	# of IPv4 prefixes	# of IPv6 prefixes
rrc00	Amsterdam	20120401.0000	435381	9078
rrc01	London	20120401.0759	405196	8684
rrc02	Paris	20081001.0759	272504	1373
rrc03	Amsterdam	20120401.0000	401302	8696
rrc04	Geneva	20120401.0759	410907	6391
rrc05	Vienna	20120401.0000	404267	8601
rrc06	Otemachi	20111001.0759	368295	0
rrc07	Stockholm	20120401.0000	407698	8374
rrc08	San Jose	20040902.0800	83509	0
rrc09	Zurich	20040204.0000	133035	0
rrc10	Milan	20120401.0000	403500	8462
rrc11	New York	20120401.0759	404834	8492
rrc12	Frankfurt	20120401.0759	409116	8708
rrc13	Moscow	20120401.0759	412170	8684
rrc14	Palo Alto	20120401.0000	408422	8551
rrc15	Sao Paulo	20120401.0759	421557	8425
rrc16	Miami	20120315.0759	407634	7826

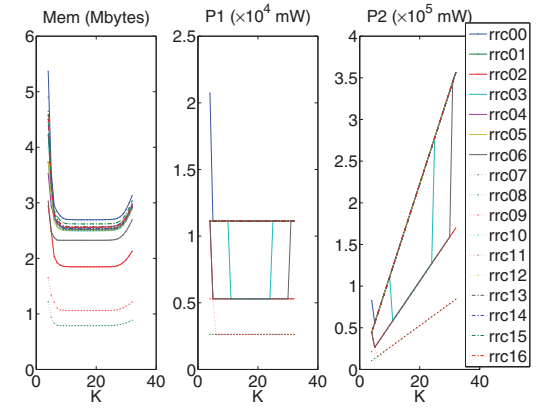
5.2 Results for NP Architecture

We increase K from 2 to the maximum prefix length (L) to assess the impact of K on the performance of the non-pipelined (NP) architecture. The results for each K are based on the optimal S_K (obtained through our solutions discussed in Section 4).

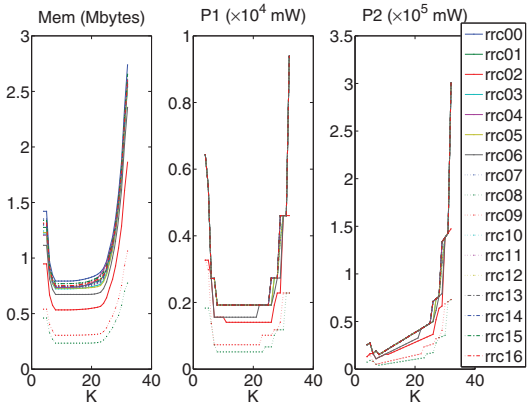
Figure 6 shows the results for IPv4 prefix sets where $L = 32$ and $K = 4, 5, \ldots, 32$. $K = 2, 3$ lead to much worse performance than the rest of Ks. Hence the

results for $K = 2, 3$ are not included in the figure for better visibility of other results. We observe that the results for the expanded trie (Figure 6(a)) and for the TBM trie (Figure 6(b)) have the similar trends:

1) Initially when K is increased from a small value, all performance metrics are getting better.
2) When K is increased from around 10 to around 25, Mem and P_1 are flat, which indicates achieving the maximum optimization. As $P_2 = K \cdot P_1$, we see the linear increase of P_2. Overall TBM tries require much less memory and power than expanded tries.
3) When K approaches L, Mem is closer to the memory requirement of a uni-bit trie. No much optimization can be done for S_K. Hence we see the increase of Mem and accordingly P_1 and P_2.



(a) Expanded trie



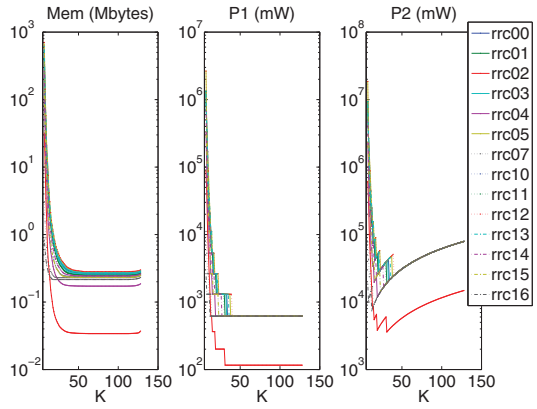
(b) Tree bitmap trie

Fig. 6. IPv4 Results for NP Architecture

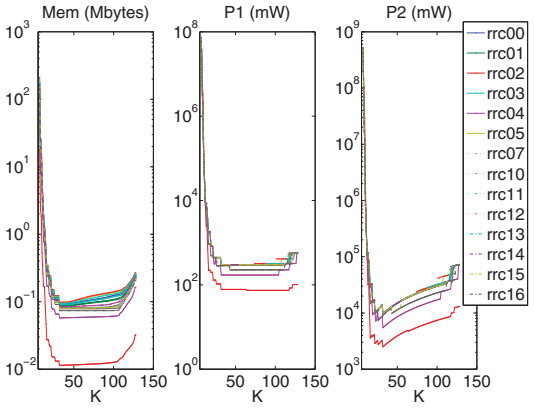
After identifying the optimal K, we obtain the corresponding optimal S_K in Table 4. The prefix sets rrc01 and rrc13 have the same results, and are listed in the same row. Similarly, rrc05, rrc10, rrc11, rrc12, rrc14 and rrc16 are listed in the same row as they share the same results. According to Table 4, we have the following findings:

1) For the expanded trie, the optimal strides that lead to the minimum memory requirement are different from those that lead to the minimum power consumption;
2) For the TBM trie, the optimal strides for large prefix sets are a series of strides of 4, which lead to both minimum memory requirement and power consumption;
3) The optimal strides of the expanded trie have a larger variation than those of the TBM trie.
4) For large prefix sets, the optimal strides for P_1 are same as those for P_2;

The second and the third findings can be explained as follows. When a TBM trie is stored in the NP architecture, the memory word width is determined by the largest stride. A TBM trie node with a small stride has to be stored in a word of the same width as a node with a larger stride. This results in memory wastage (and accordingly higher power consumption). Thus the TBM trie prefers strides of uniform value.



(a) Expanded trie



(b) Tree bitmap trie

Fig. 7. IPv6 Results for NP Architecture

Figure 7 shows the results using IPv6 prefix sets where $L = 128$ and $K = 7, 8, \ldots, 128$. The Y axis is drawn in the logarithmic scale. Since $K = 2, 3, \ldots, 6$ lead to much worse performance than the rest of Ks, the results for $K = 2, 3, \ldots, 6$ are not included in the figure for better visibility of the results. The trends for the IPv6 results are similar to those for the IPv4 results, except that when K is smaller than 16, the

TBM trie requires less memory but consumes higher power than the expanded trie. This may be due to the large word width in the TBM trie.

We also obtain the optimal S_K for IPv6 prefix sets. But due to space limitation, we do not present all the results in the paper. Table 5 lists the results for the largest IPv6 prefix set rrc00 and for the smallest non-empty IPv6 prefix set rrc02. The findings are similar to those for IPv4 results except that, the optimal strides of the TBM trie for minimum memory requirement are different from those for minimum power consumption, for both the large and the small prefix sets. The most notable result is that, the optimal strides of the TBM trie for minimum power consumption for the large prefix set (rrc00) are a series of strides of 4, which is same as that for IPv4 prefix sets.

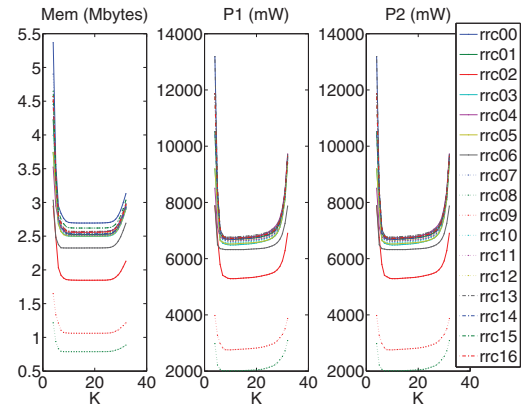
5.3 Results for SP Architecture

Figures 8 and 9 show the IPv4 and IPv6 results, respectively, about the impact of K on the performance of the simple pipelined (SP) architecture. For IPv4, $L = 32$ and $K = 4, 5, \ldots, 32$. For IPv6, $L = 128$ and $K = 7, 8, \ldots, 128$. The trends of Mem are same as those for the NP architecture. For P_1, the optimal ranges of K are narrower than those in the NP architecture. Also $P_1 = P_2$ in the SP architecture. Overall the power consumption of the SP architecture is lower than that of the NP architecture. In both architectures, the TBM trie using the optimal K achieves at least 4-fold reduction in power consumption compared with the uni-bit trie ($K = L$).

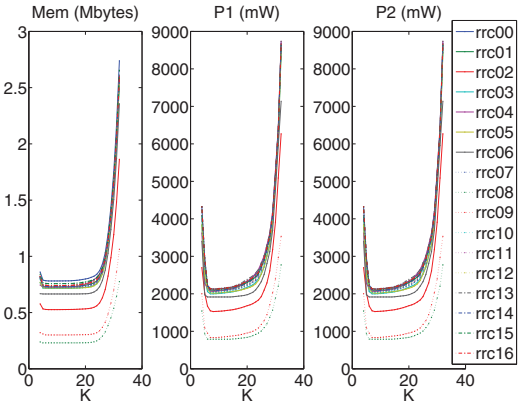
Table 6 lists the optimal strides for the seventeen IPv4 prefix sets. Table 7 lists the optimal strides for the IPv6 prefix sets rrc00 and rrc02. For the expanded trie, the optimal strides that lead to the minimum memory requirement are same as those in the NP architecture. But the optimal strides for minimum power consumption are different. For the TBM trie, the optimal strides are no longer of uniform value. This is because in a SP architecture, each stage can independently decide its word width based on the optimal stride for that stage.

5.4 Results for MBP Architecture

In a memory-balanced pipelined (MBP) architecture, $H = K + \Delta H$. In our experiments, we set $\Delta H = 1$. Figures 10 and 11 show the IPv4 and IPv6 results, respectively, about the impact of K on the performance. Compared with the results of the previous two architectures, the trends of Mem are the same while the trends of P_1 are slightly different. P_2 is similar to P_1, but not exactly the same. The sawtooth-like trends for P_1 and P_2 can be explained as follows. When K gets larger, the number of words per stage gets smaller, while the number of stages also gets larger. But a smaller number of words does not necessarily result in a smaller memory. For example, both 513 and



(a) Expanded trie



(b) Tree bitmap trie

Fig. 8. IPv4 Results for SP Architecture

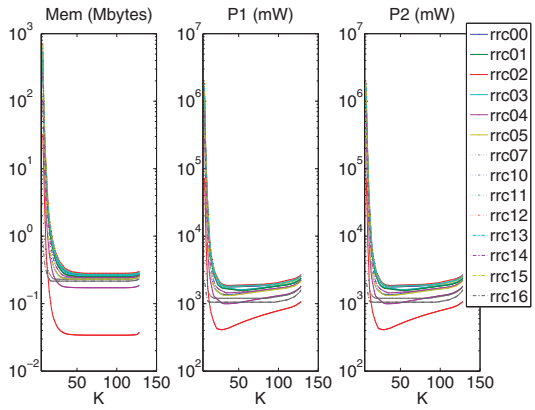
1023 words need a memory of 1024 words. As a result, there exist some cases that, with an increasing K, the power dissipation per stage is not reduced, while the number of stages is increased.

Table 8 lists the optimal strides for each IPv4 prefix set. Table 9 lists the optimal strides for IPv6 prefix sets rrc00 and rrc02. The TBM trie prefers a uniform value for the strides, which is similar to the findings for the NP architecture. But the first stride is an exception, as the word width of the first stage in a MBP architecture is independent with that in the rest of stages.

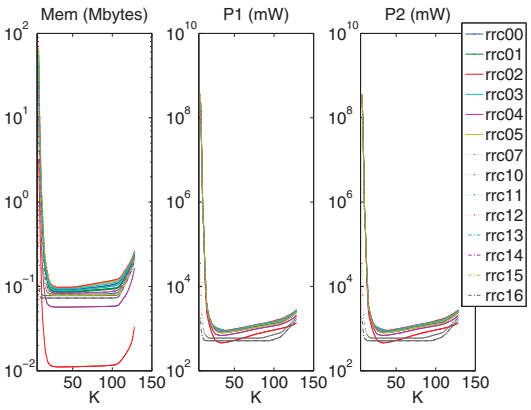
6 Related Work

6.1 Greening the Routers

Reducing the power consumption of network routers has been a topic of significant interest [5], [7], [8], [31]. Most of the existing work focuses on the system- and network-level optimizations. Chabarek et al. [7] 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

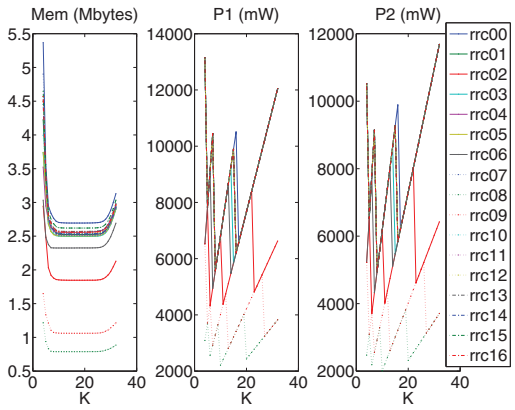


(a) Expanded trie

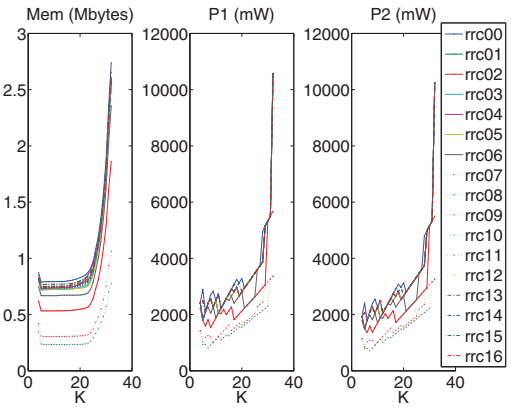


(b) Tree bitmap trie

Fig. 9. IPv6 Results for SP Architecture



(a) Expanded trie



(b) Tree bitmap trie

Fig. 10. IPv4 Results for MBP Architecture

small bursts at edge routers to facilitate sleeping and rate adaptation. These solutions can not reduce the worst-case power consumption of routers.

6.2 Power-Efficient IP Lookup Engines

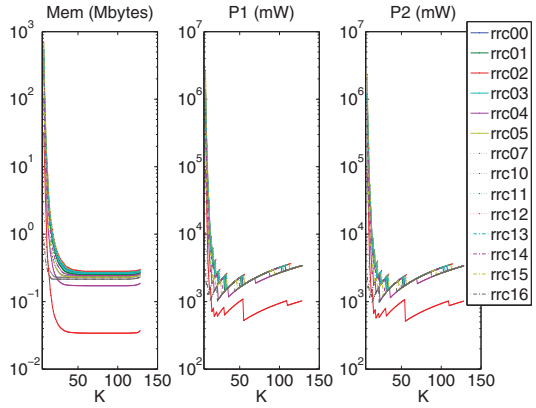
Power-efficient IP lookup engines have been studied from various aspects. However, to the best of our knowledge, there is little work done for SRAM-based IP lookup architectures. Some TCAM-based solutions [14], [32], [33] propose various schemes to partition a routing table into several blocks and perform IP lookup on one of the blocks. Similar ideas can be applied for SRAM-based multi-pipeline architectures [13]. These partitioning-based solutions for power-efficient SRAM-based IP lookup engines do not consider the underlying data structure, and are orthogonal to the solutions proposed in this paper.

Kaxiras et al. [34] 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.

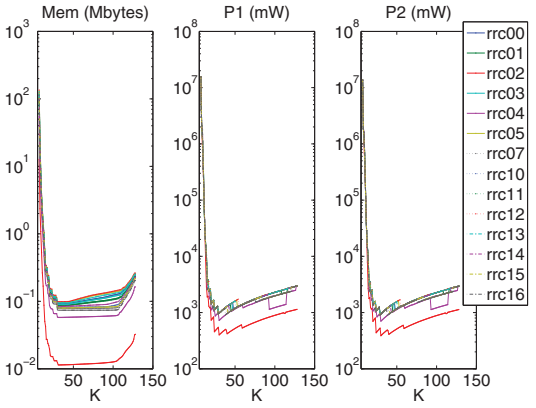
Traffic locality and rate variation have been exploited for reducing the average power consumption in IP lookup engines. Caching is employed in [16] to reduce power consumption, where some trie nodes are cached to skip the access to the deeper trie levels. In [35] clock gating is used to turn off the clock of unneeded processing engines of multi-core network processors to save dynamic power when there is a low traffic workload. In [36] a more aggressive approach of turning off these processing engines is used to reduce both dynamic and static power consumption. A finer-grained clock gating scheme is proposed in [15] to lower the dynamic power consumption of pipelined IP forwarding engines. Dynamic frequency and voltage scaling are used in [37] and [38], respectively, to reduce the power consumption of the processing engines. However, these schemes require large buffers to store the input packets so that they can determine or predict the traffic rate. The large packet buffers may result in additional high power consumption. Also, these schemes do not consider the latency for the state transition which can result in packet loss in case of burst traffic.

7 Conclusion

This paper presented a thorough study on data structure optimizations to minimize the power consumption of SRAM-based IP lookup architectures. Three



(a) Expanded trie



(b) Tree bitmap trie

Fig. 11. IPv6 Results for MBP Architecture

different architectures including the non-pipelined, the simple pipelined and the memory-balanced pipelined architectures were considered. To minimize the worst-case power consumption for each architecture, a theoretical framework was developed to determine the optimal strides for constructing multi-bit tries. Two widely-used multi-bit tries including the expand trie and the tree bitmap trie were examined. Simulation using real-life backbone routing tables including both IPv4 and IPv6 prefix sets showed that careful selection of strides in building the multi-bit tries could achieve dramatic reduction in power consumption. For each architecture and each trie algorithm, the optimal strides were different. Also the optimal strides to achieve minimum memory were different from those to achieve minimum power. We believe our methodology can be applied to other variants of multi-bit tries and can help designing more power-efficient SRAM-based IP lookup architectures.

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 and V. K. Prasanna, "Sequence-preserving parallel ip lookup using multiple sram-based pipelines," *J. Parallel Distrib. Comput.*, vol. 69, no. 9, pp. 778–789, 2009.

[3] Verizon offers U.S. 100-Gbps deployment details, "http://www.lightwaveonline.com/articles/2011/09/verizon-offers-us-100-gbps-deployment-details-129650943.html," September 2011.

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

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

[6] 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*, 2008.

[7] 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.

[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] Juniper Networks T1600 Core Router, "http://www.juniper.net."

[10] Cisco CRS-3 Router, "http://www.cisco.com."

[11] 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.

[12] 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.

[13] W. Jiang and V. K. Prasanna, "Multi-Way Pipelining for Power Efficient IP Lookup," in *Proc. Globecom*, 2008, pp. 2339–2343.

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

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

[16] L. Peng, W. Lu, and L. Duan, "Power Efficient IP Lookup with Supernode Caching," in *Proc. Globecom*, 2007.

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

[18] 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.

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

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

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

[22] 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.

[23] K. S. Kim and S. Sahni, "Efficient construction of pipelined multibit-trie router-tables," *IEEE Trans. Comput.*, vol. 56, no. 1, pp. 32–43, 2007.

[24] W. Lu and S. Sahni, "Packet forwarding using pipelined multibit tries," in *Proc. ISCC*, 2006.

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

[26] 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.

[27] 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.

[28] 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.

TABLE 4
Optimal strides for NP architecture with IPv4

Prefix set	Expanded trie			TBM trie		
	min Mem	min P_1	min P_2	min Mem	min P_1	min P_2
rrc00	3 5 4 4 2 2 2 2 1 1 1 1 1 1 2	17 4 3 3 5	17 4 3 3 5	4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4
rrc01,13	3 6 4 3 2 2 2 2 1 1 2 1 1 2	17 4 3 8	17 4 3 8	4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4
rrc02	5 4 4 3 2 2 2 2 1 1 1 1 1 1 2	17 4 3 4 4	17 4 3 4 4	4 4 4 4 4 4 4 4	3 3 3 3 3 3 3 3 3 3 2	4 5 5 5 5 5 3
rrc03	3 6 4 3 2 2 2 2 1 1 1 1 1 1 2	9 4 3 2 2 2 2 2 2 2 2	17 4 3 8	4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4
rrc04	3 6 4 3 2 2 2 2 2 2 2 2	17 4 3 8	17 4 3 8	4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4
rrc05,10-12,14,16	3 6 4 3 2 2 2 2 1 1 1 1 1 1 2	17 4 3 8	17 4 3 8	4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4
rrc06	3 6 4 3 2 2 2 2 8	16 4 2 2 8	16 4 2 2 8	4 4 4 4 4 4 4 4	4 5 5 5 5 3 5	4 5 5 5 5 3 5
rrc07	3 6 4 3 2 2 2 2 1 1 1 1 1 1 2	18 4 3 7	18 4 3 7	4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4
rrc08	3 4 3 3 3 2 2 2 2 8	16 5 3 8	16 5 3 8	4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4
rrc09	3 5 4 4 2 2 2 2 1 1 1 1 1 1 2	16 4 2 2 4 4	16 4 2 2 4 4	4 4 4 4 4 4 4 4	4 5 5 5 5 5 3	4 5 5 5 5 5 3
rrc15	3 5 4 4 2 2 2 2 1 1 1 1 1 1 2	17 4 3 8	17 4 3 8	4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4

TABLE 5
Optimal strides for NP architecture with IPv6

Prefix set	Metric	Expanded trie	TBM trie
rrc00	min Mem	2 2 4 2 2 2 2 2 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2	4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 4 4 4 4 4 4 4 4 4 2
	min P_1	10 8 6 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 4 4 4 4 4 4 4 4 5 5 5 5 4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
	min P_2	10 8 6 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 4 4 4 4 4 4 4 4 5 5 5 5 4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
rrc02	min Mem	2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
	min P_1	6 5 4 4 3 3 3 2 2 2 2 2 2 2 2 2 2 4 4 5 5 6 6 6 6 6 6 6 6 7 7	2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
	min P_2	6 5 4 4 3 3 3 2 2 2 2 2 2 2 2 2 2 4 4 5 5 6 6 6 6 6 6 6 6 7 7	4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4

TABLE 6
Optimal strides for SP architecture with IPv4

Prefix set	Expanded trie		TBM trie	
	min Mem	min $P_1 = P_2$	min Mem	min $P_1 = P_2$
rrc00	3 5 4 4 2 2 2 2 1 1 1 1 1 1 2	15 2 2 2 1 2 2 3 3	8 8 4 4 3 3 2	3 3 3 6 4 5 3 3 2
rrc01	3 6 4 3 2 2 2 2 1 1 2 1 1 2	15 2 2 2 1 2 2 3 1 2	8 8 4 4 3 3 2	3 3 3 6 4 5 3 3 2
rrc02	5 4 4 3 2 2 2 2 1 1 1 1 1 1 2	12 4 2 2 2 1 1 3 2 3	8 8 4 4 3 3 2	4 4 4 4 4 4 4 4
rrc03	3 6 4 3 2 2 2 2 1 1 1 1 1 1 2	14 3 2 2 1 2 2 2 2 2	8 8 4 4 3 3 2	3 3 3 5 5 5 4 4
rrc04	3 6 4 3 2 2 2 2 2 2 2 2	15 2 2 2 1 2 2 4 2	8 8 4 4 3 5	3 3 3 6 4 5 4 4
rrc05	3 6 4 3 2 2 2 2 1 1 1 1 1 1 2	15 2 2 2 1 2 2 3 1 2	8 8 4 4 3 3 2	3 3 3 6 4 5 2 4 2
rrc06	3 6 4 3 2 2 2 2 8	14 3 2 2 1 1 1 8	8 8 4 4 8	3 3 3 5 5 5 8
rrc07	3 6 4 3 2 2 2 2 1 1 1 1 1 1 2	15 2 2 2 1 2 2 3 1 2	8 8 4 4 2 3 1 2	3 3 3 6 4 5 3 2 1 2
rrc08	3 4 3 3 3 2 2 2 2 8	8 8 2 2 2 2 8	3 6 7 4 4 8	3 3 4 6 4 4 8
rrc09	3 5 4 4 2 2 2 2 1 1 1 1 1 1 2	8 8 4 2 1 1 4 1 1 2	3 6 7 4 4 3 3 2	3 3 4 6 4 4 4 2 2
rrc10	3 6 4 3 2 2 2 2 1 1 1 1 1 1 2	14 3 2 2 1 2 3 3 2	8 8 4 4 3 3 2	3 3 3 5 5 5 3 3 2
rrc11	3 6 4 3 2 2 2 2 1 1 1 1 1 1 2	15 2 2 2 1 2 1 2 2 3	8 8 4 4 2 3 1 2	3 3 3 6 4 5 3 2 3
rrc12	3 6 4 3 2 2 2 2 1 1 1 1 1 1 2	15 2 2 2 1 2 2 2 2 2	8 8 4 4 3 3 2	3 3 3 6 4 5 3 3 2
rrc13	3 6 4 3 2 2 2 2 1 1 2 1 1 2	15 2 2 2 1 2 2 2 2 2	8 8 4 4 2 3 3	3 3 3 6 4 5 4 4
rrc14	3 6 4 3 2 2 2 2 1 1 1 1 1 1 2	15 2 2 2 1 2 2 3 1 2	8 8 4 4 3 3 2	3 3 3 6 4 5 3 3 2
rrc15	3 5 4 4 2 2 2 2 1 1 1 1 1 1 2	15 2 2 2 1 2 1 2 2 1 2	8 8 4 4 3 3 2	3 3 3 6 4 5 3 3 2
rrc16	3 6 4 3 2 2 2 2 1 1 1 1 1 1 2	15 2 2 2 1 2 1 2 2 3	8 8 4 4 3 3 2	3 3 3 6 4 5 3 3 2

[29] W. Jiang and V. K. Prasanna, "Architecture-aware data structure optimization for green ip lookup," in *HPSR*, 2010, pp. 113–118.
[30] RIS Raw Data, "http://data.ris.ripe.net."
[31] I. Keslassy, S.-T. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard, and N. McKeown, "Scaling internet routers using optics," in *Proc. SIGCOMM*, 2003, pp. 189–200.
[32] 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.
[33] W. Lu and S. Sahni, "Low power tcams for very large forwarding tables," *IEEE/ACM Trans. Netw.*, vol. 18, no. 3, pp. 948–959, 2010.
[34] S. Kaxiras and G. Keramidas, "IPStash: a set-associative memory approach for efficient IP-lookup," in *INFOCOM*, 2005, pp. 992–1001.
[35] 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.
[36] R. Kokku, U. B. Shevade, N. S. Shah, M. Dahlin, and H. M. Vin, "Energy-Efficient Packet Processing," in *http://www.cs.utexas.edu/users/rkoku/RESEARCH/energy-tech.pdf*, 2004.
[37] A. Kennedy, X. Wang, Z. Liu, and B. Liu, "Low power architecture for high speed packet classification," in *Proc. ANCS*, 2008, pp. 131–140.
[38] M. Mandviwalla and N.-F. Tzeng, "Energy-efficient scheme for multiprocessor-based router linecards," in *Proc. SAINT*, 2006.

Weirong Jiang received his B.S. (2004) in automation and M.S (2006) in control science and engineering from Tsinghua University,

TABLE 7
Optimal strides for SP architecture with IPv6

Prefix set	Metric	Expanded trie	TBM trie
rrc00	min Mem	2 2 4 2 2 2 2 2 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	4 6 4 5 6 3 4 4 4 4 4 4 4 4 4 3 3 3 3 3 3 3 3 3 4 3 3 3 4 4 3 3 4 4 2
	min P_1=P_2	4 6 4 7 4 3 2 2 2 2 2 2 2 2 3 3 1 4 4 4 3 4 4 4 4 2 2 2 2 2 2 4 4 4 4 4 4 4 4 4 2	3 3 3 4 4 4 4 3 4 3 3 3 3 4 2 3 3 4 4 3 3 3 3 3 3 3 3 4 4 3 3 3 3 3 3 3 3 3 3 3 2
rrc02	min Mem	2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	2 4 4 4 4 4 4 6 4 4 4 4 3 3 3 3 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4
	min P_1=P_2	6 6 5 3 2 1 4 3 2 4 4 5 3 4 4 4 4 5 5 5 5 5 5 5 5 5 5 6 8	4 4 4 4 4 4 3 5 3 2 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4

TABLE 8
Optimal strides for MBP architecture with IPv4

Prefix set	Expanded trie			TBM trie		
	min Mem	min P_1	min P_2	min Mem	min P_1	min P_2
rrc00	3 5 4 4 2 2 2 2 1 1 1 1 1 1 2	13 4 3 2 2 2 3 3	13 4 3 2 2 2 3 3	16 4 4 4 4	12 4 4 4 4 4	12 4 4 4 4 4
rrc01	3 6 4 3 2 2 2 2 1 1 2 1 1 2	13 4 3 2 2 2 3 3	13 4 3 2 2 2 3 3	16 4 4 4 4	16 4 4 4 4	16 4 4 4 4
rrc02	5 4 4 3 2 2 2 2 1 1 1 1 1 1 2	16 4 2 2 4 4	16 4 2 2 4 4	16 4 4 4 4	4 4 4 4 4 4 4 4	18 3 3 3 3 2
rrc03,10	3 6 4 3 2 2 2 2 1 1 1 1 1 1 2	12 4 2 2 2 2 4 4	12 4 2 2 2 2 4 4	16 4 4 4 4	16 4 4 4 4	16 4 4 4 4
rrc04	3 6 4 3 2 2 2 2 2 2 2 2	12 4 2 2 2 2 5 3	12 4 2 2 2 2 5 3	16 4 4 4 4	16 4 4 4 4	16 4 4 4 4
rrc05	3 6 4 3 2 2 2 2 1 1 1 1 1 1 2	12 4 2 2 2 2 3 5	12 4 2 2 2 2 3 5	16 4 4 4 4	16 4 4 4 4	16 4 4 4 4
rrc06	3 6 4 3 2 2 2 2 8	12 4 2 2 2 2 8	12 4 2 2 2 2 8	16 4 4 4 4	16 4 4 4 4	16 4 4 4 4
rrc07,12	3 6 4 3 2 2 2 2 1 1 1 1 1 1 2	13 4 3 2 2 2 3 3	13 4 3 2 2 2 3 3	16 4 4 4 4	16 4 4 4 2 2	16 4 4 4 2 2
rrc08	3 4 3 3 3 2 2 2 2 8	3 4 3 3 3 2 2 2 2 8	3 4 3 3 3 2 2 2 2 8	8 4 4 4 4 4 4	8 4 4 4 4 4 4	8 4 4 4 4 4 4
rrc09	3 5 4 4 2 2 2 2 1 1 1 1 1 1 2	3 5 4 4 2 2 2 2 2 1 2 1 2	12 5 3 2 2 4 4	8 4 4 4 4 4 4	1 3 4 4 4 4 4 4 4	16 4 4 4 4
rrc11,14	3 6 4 3 2 2 2 2 1 1 1 1 1 1 2	13 4 3 2 2 2 3 3	13 4 3 2 2 2 3 3	16 4 4 4 4	16 4 4 4 4	16 4 4 4 4
rrc13	3 6 4 3 2 2 2 2 1 1 2 1 1 2	13 4 3 2 2 2 3 3	13 4 3 2 2 2 3 3	16 4 4 4 4	12 4 4 4 4 4	12 4 4 4 4 4
rrc15	3 5 4 4 2 2 2 2 1 1 1 1 1 1 2	13 4 3 2 2 2 3 3	13 4 3 2 2 2 3 3	16 4 4 4 4	16 4 4 4 2 2	16 4 4 4 2 2
rrc16	3 6 4 3 2 2 2 2 1 1 1 1 1 1 2	13 4 3 2 2 2 3 3	13 4 3 2 2 2 3 3	16 4 4 4 4	12 4 4 4 4 4	12 4 4 4 4 4

TABLE 9
Optimal strides for MBP architecture with IPv6

Prefix set	Metric	Expanded trie	TBM trie
rrc00	min Mem	2 2 4 2 2 2 2 2 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 4 4 4 4 4 4 4 4 4 4 4 2
	min P_1	10 8 6 2 2 2 2 2 2 2 2 2 2 2 2 2 4 4 4 4 5 5 5 5 5 5 5 4 5 4 4 4 4 4	16 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
	min P_2	10 8 6 2 2 2 2 2 2 2 2 2 2 2 2 2 4 4 4 4 5 5 5 5 5 5 5 4 5 4 4 4 4 4	16 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
rrc02	min Mem	2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
	min P_1	2 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4	12 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
	min P_2	2 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4	12 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4

Beijing, China, and Ph.D. (2010) in Computer Engineering from University of Southern California, Los Angeles. His research is on parallel algorithm and architecture design for high-performance low-power packet processing in Internet infrastructure. His research interests also include network security, reconfigurable computing, network virtualization, data mining, and wireless networking.

Viktor K. Prasanna is Charles Lee Powell Chair in Engineering in the Ming Hsieh Department of Electrical Engineering and Professor of Computer Science at the University of Southern California. His research interests include High Performance Computing, Parallel and Distributed Systems, Reconfigurable Computing, and Embedded Systems. He received his BS in Electronics Engineering from the Bangalore University, MS from the School of Automation, Indian Institute of Science and Ph.D in Computer Science from the Pennsylvania State University. He is the Executive Director of the USC-Infosys Center for Advanced Software Technologies (CAST) and is an Associate Director of the USC-Chevron Center of Excellence for Research and Academic Training on Interactive Smart Oilfield Technologies (Cisoft). He also serves as the Director of the Center for Energy Informatics at USC. He served as the Editor-in-Chief of the IEEE Transactions on Computers during 2003-06. Currently, he is the Editor-in-Chief of the Journal of Parallel and Distributed Computing. He was the founding Chair of the IEEE Computer Society Technical Committee on Parallel Processing. He is the Steering Co-Chair of the IEEE International Parallel and Distributed Processing Symposium (IPDPS) and is the Steering Chair of the IEEE International Conference on High Performance Computing (HiPC). Prasanna is a Fellow of the IEEE, the ACM and the American Association for Advancement of Science (AAAS). He is a recipient of the 2009 Outstanding Engineering Alumnus Award from the Pennsylvania State University.