EaseCAM: An Energy And Storage Efficient TCAM-based Router Architecture for IP Lookup

V C Ravikumar and Rabi Mahapatra
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
{vcr,rabi}@cs.tamu.edu

Laxmi N Bhuyan
Department of Computer Science
University of California
Riverside, CA 92521
bhuyan@cs.ucr.edu

Abstract—Ternary Content Addressable Memories (TCAMs) have been emerging as a popular device in designing routers for packet forwarding and classifications. Despite their promise on high-throughput, the use of large TCAM array is prohibitive due to its excessive power consumption and lack of scalable design schemes. This paper presents a TCAM-based router architecture that is energy and storage efficient. We introduced new prefix aggregation and expansion techniques to compact the effective TCAM size in a router. Pipelined and paging schemes are employed in the architecture to activate a limited number of entries in the TCAM array during an IP lookup. The new architecture provides low power, fast incremental updating, and fast table look-up. Heuristic algorithms for page filling, fast prefix update, and memory management are also provided. Results have been illustrated with two real-life large routers (bbnplanet and attcanada) to demonstrate the effectiveness of our approach.

Keywords: router, prefixes, TCAMs, IP Lookup, partition, compaction, page table

I. INTRODUCTION

Internet protocol (IP) lookup forms a bottleneck in packet forwarding in modern IP routers because the lookup speed is unable to catch up with the increase in link bandwidth. The Ternary Content Addressable Memories (TCAMs) have been emerging as viable devices for designing high throughput forwarding engines on routers. They store don’t care states in addition to 0s and 1s, and search the data (IP address) in a single clock cycle. This property makes TCAMs particularly attractive for packet forwarding and classifications. Despite these advantages, the use of large TCAM arrays is prohibitive due to large power consumptions and lack of scalable design schemes.
The high density TCAMs, available in the market today, consumes 12-15W/chip when the entire memory is enabled. In order to support the IP prefixes that are increasing at significant rate, vendors use 4 to 8 TCAM chips. More chips would also be required to handle filtering and packet classification in addition to the IP lookup. The power consumption resulting in using large number of chips not only increases the cooling costs but also limits the router design to fewer ports. Recent research in reducing the power consumption of the TCAM has been discussed in [8][10]. The power consumption increases linearly with the increase in the number of entries and bits in a TCAM. Hence, the techniques proposed for compacting the routing table will result in substantial reduction in power. Liu [9] has presented a novel technique to eliminate the redundancies in the routing table. However, we show that this technique takes excessive time for updating because this is based on Espresso-II minimization algorithm, whose complexity increases exponentially with the size of the original routing table. Thus, the main motivation of this paper is to come up with a TCAM router architecture that consumes low power and is suitable for incremental updating needed in modern IP routers. Additionally, we also minimize the memory size and look-up delay.

We propose a pipelined architecture that can achieve the above goals through further compaction of the active region of the routing table in TCAM architecture. In this paper we introduce the idea of prefix aggregation and prefix expansion for TCAM, which can reduce the number of entries for comparison in a TCAM. Together with the prefix aggregation and overlapping properties, we select a limited number of pages during IP look-up instead of energizing the entire module. The present TCAM vendors provide mechanisms to enable a chunk of TCAM, much smaller compared to the entire TCAM.
We exploit this technology to achieve an upper bound on the power consumption in the TCAM based router. Based on statistical analysis of the current IP routing tables, we present a 2-level paged TCAM architecture. Detailed analysis and design tradeoffs are presented by varying the sub-prefix length. We also propose page filling heuristic to improve memory utilization due to paging. The concept of bucket-TCAM has been used to isolate storing of rarely incoming IP prefixes and to find optimal page size in the proposed architecture. An efficient memory management scheme is presented for updating the routing entries. Finally, we derive empirical equations for memory requirement and power consumption for the proposed architecture.

Case studies were made using the statistics from bbnplanet and attcanada routers to demonstrate the effectiveness of the proposed TCAM architecture. It is shown that less than 1% of power is consumed when compared to energizing the full-size TCAM implementation. Considerable saving is also noticed in memory size and look-up delay.

The paper makes the following significant contribution.

- Introduced prefix-aggregating technique to substantially reduce TCAM size.
- It presents a two-level pipelined architecture based on which selected pages and modules can be activated during a table look-up process.
- Efficient paging and memory management techniques are derived to improve TCAM utilization and update operations.
- Through analysis and simulation, it demonstrates substantial reduction in power and memory size when the proposed architecture is used.
The rest of the paper is organized as follows. Section II introduces new compaction techniques based on some of the prefix properties that are critical for this work. Section III discusses the proposed architecture for the forwarding engine and tackles the problem of optimizing power and memory. The update mechanism and memory management have been addressed in Section IV. Section V illustrates the results and case studies based on statistics from two real routing tables. A brief summary on related work is given in Section VI. In Section VII, we conclude this work.

II. PREFIX COMPACTION

The purpose of delving into the prefix properties is three fold: 1) To further increase the compaction of the routing table in addition to using the existing techniques. 2) To come up with an upper bound on minimization time lest it become a bottleneck in the routing lookup. 3) To derive an upper bound on the power consumption during a lookup process.

The previous attempts to reduce the routing table size in TCAM using prefix properties have been achieved using the property of Pruning and Mask extension [9]. Pruning achieves compaction by storing only one of the many overlapping prefixes that have the same next-hop. Two prefixes are overlapping if one is an enclosure of the other. Let $P_i$ denote the prefix $P_i$ and $|P_i|$ denotes the length of prefix $P_i$ then $P_i \in P$ is called *enclosure* of $P_j$, if $P_j \in P$, $j \neq i$ and $|P_i| < |P_j|$, such that $P_i$ is a sub-prefix of $P_j$. If $P_i$ and $P_j$ have the same next hop then they can be represented by $P_i$. Thus, a set of overlapping prefixes $\{P_1, P_2, P_3, \ldots, P_n\}$, such that $|P_1| < |P_2| < |P_3| < \ldots < |P_n|$, having the same next hop can be replaced with a single entry $P_1$. If an update deletes the entry $P_1$, the set of overlapping prefixes $\{P_1, P_2, \ldots, P_n\}$ should be represented by the entry $P_2$. When an update adds a new
entry $P_i$, such that $|P_i| < |P_1| < |P_2| \ldots < |P_n|$, then the existing entry $P_1$ is replaced with $P_i$. However, if the new entry $P_i$ arrives, such that, $|P_i| > |P_1|$, then no changes are made to the routing table, except for the updating of set of overlapping prefixes. The *Mask extension* property logically minimizes two or more prefixes to a minimal set, if these prefixes have the same next hop. The logic minimization is an NP-complete problem and has been addressed using the Espresso-II algorithm [4]. Using Espresso-II, the prefixes $\{P_1, P_2, P_3, \ldots P_n\}$ are minimized to $\{P'_1, P'_2, P'_3, \ldots P'_m\}$, such that $m \leq n$.

While the *Pruning* and *Mask extension* technique together compacts about 30-45% of the routing table, one need to investigate the overhead in using these approaches for real time updates. The time taken for pruning is bounded and is independent of the size of the router. However, for mask extension, the logic minimization algorithm using Espresso-II has an exponential runtime with input size. Based on the technique discussed in [9], the input to Espresso-II algorithm can be as high as 15,146 for the *attcanada* router. Thus, runtime for such a large size input data can take several tens of minutes and is very expensive for incremental updates. To tackle this problem, we use another property called *prefix aggregation* that will derive an upper bound on input data size before minimization is applied. The minimization of the prefixes based on this property is time bounded.

<table>
<thead>
<tr>
<th>IP Address</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>129.66.6.0/24</td>
<td>4.0.6.142</td>
</tr>
<tr>
<td>129.66.8.0/24</td>
<td>4.0.6.142</td>
</tr>
<tr>
<td>129.66.12.0/24</td>
<td>4.0.6.142</td>
</tr>
<tr>
<td>129.66.20.0/24</td>
<td>4.0.6.142</td>
</tr>
<tr>
<td>129.66.21.0/24</td>
<td>4.0.6.142</td>
</tr>
<tr>
<td>129.66.30.0/23</td>
<td>4.0.6.142</td>
</tr>
<tr>
<td>129.66.31.0/24</td>
<td>4.0.6.142</td>
</tr>
<tr>
<td>129.66.32.0/19</td>
<td>4.0.6.142</td>
</tr>
<tr>
<td>129.66.34.0/24</td>
<td>4.0.6.142</td>
</tr>
<tr>
<td>129.66.47.0/24</td>
<td>4.0.6.142</td>
</tr>
<tr>
<td>129.66.48.0/24</td>
<td>4.0.6.142</td>
</tr>
<tr>
<td>129.66.64.0/18</td>
<td>4.0.6.142</td>
</tr>
<tr>
<td>129.66.88.0/24</td>
<td>4.0.6.142</td>
</tr>
<tr>
<td>129.66.95.0/24</td>
<td>4.0.6.142</td>
</tr>
<tr>
<td>129.66.111.0/24</td>
<td>4.0.6.142</td>
</tr>
<tr>
<td>129.66.128.0/22</td>
<td>4.0.6.142</td>
</tr>
<tr>
<td>129.66.132.0/24</td>
<td>4.0.6.142</td>
</tr>
<tr>
<td>129.66.172.0/24</td>
<td>4.0.6.142</td>
</tr>
</tbody>
</table>

Fig.1. Sample trace of routing table from *bbnplanet*
A. Prefix Aggregation

In this sub section we introduce a new property of the prefixes called *prefix aggregation*. Figure 1 represents a portion of the routing table dump taken from the bbnplanet router. It represents all the prefixes in the routing table starting with 129.66 and having prefix length > 16 and ≤ 24. We call 129.66, as the largest common sub-prefix (LCS) for the set of prefixes on the traces in Figure 1. We define LCS as the sub-prefix whose length is the nearest multiple of 8 such that |S_i|<|P_i|, where S_i is the LCS of prefix P_i [LCS(P_i)]. However, if the prefixes under consideration are such that |P_i| > w_1 \ \forall \ \ P_i, \text{ where } w_1 \text{ is an integer}, then LCS of P_i is \((p_i_1,p_i_2...p_i_w)\) if \(|P_i| \leq (w_i/8 + 1)^8\) (p_{ik} represents the k\textsuperscript{th} bit of the prefix P_i). Statistical observations show that if the prefixes are grouped on basis of their LCS, they correspond to the same next-hop with a very few exceptions. Interestingly, it is also observed that prefixes corresponding to different LCS usually do not have the same next-hop. According to these observations, we may state that nearly same degree of compaction is achieved when minimization is applied on to the groups of prefixes partitioned by their LCS with respect to when those are not partitioned, Table I gives an indication about the extent of compaction achieved by prefix aggregation and the maximum compaction possible in the two routers. Thus based on the prefix statistics in the core routers, we can presume that the set of largest common sub-prefixes could be treated as mutually exclusive. Notice that our algorithm does not produce maximum prefix compaction, but it saves considerable compaction time by limiting the input data size to the Espresso algorithm used during compaction. As a result, the proposed technique is more suitable to incremental (on-line) updating as
opposed to the technique in [9]. More detailed explanation on this property is given later in this section.

The use of prefix aggregation property provide an upper bound on the number of possible prefixes that can be given as input to the minimization algorithm, without significantly affecting the total amount of compaction. These prefix sets are always of the largest sized input set and prefixes in each set correspond to the same LCS. Let the LCS for prefix $P_i = (p_{i1}, p_{i2}...p_{i|P_i|})$ be represented as $S = (p_{i1}, p_{i2}...p_{i(|S|+8)})$. Then the aggregated set $P$ is a set of prefixes $\forall P_i \in P$ such that the maximum common sub-prefix of $P_i$ is $S$.

**Observation II.1:** The maximum number of prefixes with the same largest common sub-prefix in a prefix set is 256.

**Proof:** Let $S$ be the largest common sub-prefix for the set of prefixes $P_i$. The Classless Inter-domain Routing (CIDR) addressing was introduced so that if $S$ covers all ranges of prefixes $P_i$, such that LCS ($P_i$) = $S$ and, $|S| < |P_i| \leq (|S|+8)$. Then new subnets can be added with prefix $P_i$ that will result in networks having smaller number of hosts. Thus to find the maximum possible number of such prefixes that can be added, we assign all possible combinations of $P_i$, such that $|P_i| = (|S|+8)$. This will ensure that it is not possible to add

<table>
<thead>
<tr>
<th>Router</th>
<th>Total Prefixes</th>
<th>Max Prefix Compaction</th>
<th>Prefix Aggregation based Compaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATT-Canada</td>
<td>112412</td>
<td>54476</td>
<td>57837</td>
</tr>
<tr>
<td>Bbn-planet</td>
<td>124538</td>
<td>69646</td>
<td>71500</td>
</tr>
</tbody>
</table>

**TABLE I.** Comparision of prefix minimization using prefix aggregation property in attcanada and bbnplanet router

$\begin{array}{|c|c|c|c|}
\hline
\text{Router} & \text{Total Prefixes} & \text{Max Prefix Compaction} & \text{Prefix Aggregation based Compaction} \\
\hline
\text{ATT-Canada} & 112412 & 54476 & 57837 \\
\hline
\text{Bbn-planet} & 124538 & 69646 & 71500 \\
\hline
\end{array}$
any Pi such that |S|< |Pi| < (|S|+8). Thus we can have a total of $2^{(|S|+8) - |S|} = 256$

B. Prefix Expansion

The concept of prefix expansion for IP lookup has been discussed using software approaches as in [6]. We adopt this property here to further compact the routing table. The Prefix Expansion property can be represented mathematically as follows. If $P_i$ represent a prefix, such that $|P_i|$ is not a multiple of 8, then the prefix expansion property expands $P_i$ to $P_i.X^m$ such that, $X=$don’t care and $m = 8-(|P_i| \mod 8)$. The operator “.” represents the concatenation operation.

Based on Observation II.1, an input prefix sets for logic minimization can have maximum size 256, corresponds to same LCS and will have a bounded runtime. For each prefix $P_i \in P$, that has a LCS $S$, $|P_i|$ may not be a multiple of 8. However Espresso-II algorithm will provide more compaction if all prefixes $P_i$’s in the set are of same length. Hence to increase the compaction, we expand the prefixes that do not confirm to these requirements.

The run time of the Espresso-II algorithm is not only bounded by the number of inputs but also by the bit length (width) of the input. From previous discussions we observe that the input set to the Espresso algorithm is the set of prefixes that have the same LCS. Thus the largest common sub-prefix that is a part of all the prefixes is redundant to the Espresso-II algorithm. Thus it would be useful, to only give the varying bits of the set containing the largest common sub-prefix, which is the least significant 8 bits of each prefix after prefix expansion. Thus for each $P_i \in P$ that has the same LCS $S$, ...
the optimal input set of prefixes to the Espresso algorithm to maximize the compaction with an upper bound on runtime is the set $P'$, Such that $\forall P'_i \in P'$  

$N \times q_i$ for $|P_i| \neq m*8$, m is an integer  

$N$ for $|P_i| = m*8$, m is an integer  

where, $N=(p_{i_1}, p_{i_2}, ..., p_{i_{k-1}})$  

$$k=8*\frac{(|P_i|-1)}{8}+1$$  

$$q_i=8-|P_i|$$  

**C. Overlapping prefixes for an IP address**

The maximum number of overlapping prefixes for a given IP address indicates the minimum number of prefixes one needs to search during a lookup operation. Thus, if we search a bounded numbers of prefixes during any lookup operation, the power consumption will be limited to those entries that are enabled during the search process. In the following section, we show that during a search process, only a bounded number of prefixes ($256*3$) will be compared. These prefixes will contain the set of overlapping prefix for any IP address.

**Observation II.2: The total number of overlapping prefixes for a given IP address is $\leq 25$.**

**Proof:** Consider the incoming IP address $I = (i_1i_2 .. i_{32})$. For every $I$, we can generate a $P_{il}$, such that, $1 \leq l \leq 32$, and $P_i = (i_1, i_2, ..., i_l)$. Since there are 32 such values of $l$, we can have 32 such possible overlapping prefixes of $I$. Since in today’s routers there are no prefixes of length < 8, the maximum number of possible overlapping prefixes for $I$ is $32-7=25$. 
III. DESIGN OF ROUTER ARCHITECTURE

A. Deriving a TCAM Architecture

In the proposed architecture, compaction of the routing table is achieved by exploiting the techniques presented here. Notice that this approach is different from one in [9] in order to accommodate fast incremental updates. We adopt 2-level routing lookup architecture as shown in Figure 2. The 1st level contains $w_1$-bit sub-prefixes which are compared with $w_1$ most significant bits of incoming IP address. If there is a match in the 1st level, it enables corresponding region in the 2nd level. The size of this selected region varies from router to router. This region is called as segment in Figure 2. This would mean that the worst-case power consumption is decided by the size of the largest segment, which is not bounded. For large sized routers the segment size could be very large. For example, with $w_1 = 8$, the size of the largest segment in bbnplanet router is 7580 entries.
Choice of the number of bits in the 1\textsuperscript{st} level affects the compaction and performance of the proposed architecture. Let us assume that the 1\textsuperscript{st} level entries do not contain any don’t care bits. This will ensure that only a unique segment in the 2\textsuperscript{nd} level will be selected. Depending on the width of $w_1$ at the 1\textsuperscript{st} level, the size of the TCAM arrays and total memory requirements will vary and which will affect the performance.

The compaction technique is applied as follows. First we consider all the prefixes that are greater than $w_1$ bits from routing table traces. Using the property of prefix overlapping we remove all the redundant entries in the routing table. We then apply the property of prefix aggregation on these entries to form sets of prefixes having largest common sub-prefix. Each of these sets are expanded using the prefix expansion technique and are subjected to minimization using the Espresso-II algorithm. Then the minimized prefixes are stored in the 2\textsuperscript{nd} level TCAM that are of length $32-w_1$.

![Fig.3 Total number of entries after compaction](image)

Figure 3 shows the number of entries in the TCAM after compaction for \texttt{bbnplanet} and \texttt{attcanada} routers, using the compaction technique while varying the value of $w_1$. For the value of $w_1$ between 8 and 18, the total compaction decreases with increasing $w_1$ for both the routers. This is because as $w_1$ increases the input to the Espresso-II algorithm decreases (prefixes $< w_1$ are not given as input to the Espresso
algorithm). However for \( w_1 > 18 \), we see that the number of compacted entries is decreasing with increasing \( w_1 \). This is because of the fact that the number of prefixes available for compaction gets reduced in the 2\textsuperscript{nd} level TCAM. At this point the benefit of compaction is not noticeable.

In this section we present the detailed architecture of the TCAM based packet forwarding architecture. The discussions on various components of the architecture and design parameters are presented. We introduce the heuristics based on prefix properties to obtain a bound to power consumption based on the active TCAM entries during a single lookup operation. Heuristics to store the pages efficiently in the TCAM for high memory utilization is also discussed. Lastly, we also introduce an empirical model for power and memory requirements based on the parameters involved in the architectural design.

\textit{B. Paged TCAM architecture}

One of our goals is to estimate the total memory requirements in TCAM arrays and maximum power consumption in the lookup process for different values of prefix lengths \((w_1)\). In order to determine the maximum power consumption per IP lookup, we need to determine maximum number of TCAM entries enabled during any lookup process. These estimations depend on the design parameters of the proposed architecture.

The implementation scheme to realize segmentation in TCAM array is non-trivial. In order to implement the segmentation concept, we consider a paging scheme for actual implementation of the segments. Though the paging scheme for TCAMs is used in [8] and [10], we handle the paging differently due to architectural differences. We now present the heuristics to efficiently store bounded number of enabled entries for less power consumption.
Determining bound on page size

We propose to select bounded number of pages in a segment during IP lookup instead of selecting all the pages using prefix aggregation technique. The prefixes having the same largest common sub-prefix are stored in the pages that have the same page ID’s, and are represented by their LCS values. Then the maximum number of pages enabled during the lookup process is bounded (Observation III.1). In the bbnplanet router, the number of entries in each page can range from minimum of 1 to the maximum of 256. Though this technique can achieve a bound on the number of entries in a page, it may not be the best solution due to its low memory utilizations.

Observation III.1: When the pages are created using aggregation technique, the maximum number of pages enabled that contains the set of overlapping prefixes during a lookup process is \( \leq 3 \).

Proof: From Observation II.2 we know that the total number of possible overlapping prefixes for a given IP address is \( \leq 25 \). One needs to find out the maximum number of pages that will be searched based on the aggregation technique. This is the set of pages containing the set of overlapping prefixes. Since we have considered the number of bit-lines in the 1st level \((w_1) \geq 8\), the total number of overlapping prefixes is \( \leq 24 \). We now prove the bound for the value of \( w_1=8 \), which has the largest number of overlapping prefixes. The prefix aggregation technique groups the prefixes such that they have a common prefix, which is a multiple of 8 for \( w_1=8 \). Hence there are three such groups possible with maximum common prefix lengths of 8, 16, and 24. Thus the total number of pages containing overlapping prefixes cannot exceed 3 based on the aggregation technique.
The upper bound on the number of entries that need to be searched during an IP lookup cannot exceed $256 \times 3$.

The statistics from the bbnplanet router show that the segment sizes vary depending on the value of $w_1$. Thus if the page size is large, having separate pages for the smaller sized segments would result in a wastage of space. Similarly if the segment sizes are large having small sized pages will increase the number of page ID’s and hence the memory utilization. Hence a heuristic is proposed to compute the right page size based on $w_1$ that would minimize the memory consumption. In the next section we describe the heuristics to obtain the optimal page size, among all the possible page sizes that are possible. Since the sets of prefixes formed using the aggregation technique cannot exceed 256 entries, the maximum size of the page cannot exceed 256 entries. We refer to the term $\beta$ to indicate to the page size. Our goal is to find the optimal value of $\beta$ ($\beta_{w_1}$) for all possible $w_1$, which will optimizes memory.

Page Filling Heuristics

Though the aggregation technique gives an upper bound on the power consumption, it may increase the TCAM’s size due to the memory under utilization. We present a heuristics based on the maximal covering of prefixes to increase memory utilization. Let a cube represent a single entry for a set of prefixes and covering represents the set of cubes that cover all the prefixes that have the same LCS($P_i$). Our goal is to find the minimal set of such cubes that combine prefixes $P_i$ having the same $|S_i|$. Since $|S_i|$ can have a maximum of 3 values for any value of $w_1$ in the proposed architecture, there can be a maximum of 3 overlapping cubes for a given IP address if the cubes in each cover are made non-overlapping. This ensures that the maximum number
of active or enabled entries cannot exceed 256*3. This will be true if all the prefixes in the cube $C_i$ can be arranged in pages so that the total number of entries in all pages does not exceed 256. Each of these cubes will represent the page ID for the page containing the prefixes it overlaps. Further, we introduce a parameter $\gamma$, called fill factor, which represents the fraction of a page filled during reconfiguration process that covers prefixes with different LCS values. The page-filling algorithm is presented Figure 4 (a) & (b). The page-filling algorithm tries to fill the entries into the pages, each of size $\beta$, efficiently by trying to find the minimal number of cubes that are non-overlapping and cover all the prefixes using MinimalCoverSet. The StorePage algorithm ensures that no page has more than $\beta*\gamma$ entries.

**FillPages**($P, w_1, \beta, \gamma, C'$)
// $P_1 \in P$, such that $|P_1| > w_1$, $\forall P_1$
Let $Q_{w_1} = p_1p_2p_3...p_{w_1}$
Page = 0
$C_{\text{max}} = 0$
For all $P_1 \in P$ with same $|\text{LCS}(P_1)|$ and $Q_{w_1}$
For $|\text{LCS}(P_i)| <= l <= (|\text{LCS}(P_i)|/8 + 1) * 8$
    $C_i = \text{MinimalCoverSet}(P, \gamma)$
EndFor
For all $P_1 \in P$ covered by $C_i$
    If ($C_i$ covers $P_i$'s having same $\text{LCS}(P_i)$)
        Page += StorePage($C_i, P, \beta, \gamma$)
    Else
        Page += StorePage($C_i, P, \beta, 1$)
    EndFor
If ($C > C_{\text{max}}$)
    $C_{\text{max}} = C$
EndFor
$C' = C' + C$
EndFor

**StorePage**($C_i, P, \beta, \gamma$)
While $P \neq 0$
    Create New Page with Page ID $C_i$
    Entry = 0
    While Entry < $\beta*\gamma$
        AddToPage($P_i$)
        $P = P - \{P_i\}$
        Entry++
    EndWhile
EndFor

Figure 4 (a): Page Filling heuristics

Figure 4 (b): Heuristics for storing prefixes into pages
C. Comparator, Page tables, pages and bucket

Figure 5 represents detailed architecture of the proposed approach. The 1\textsuperscript{st} level lookup consists of comparators and page tables, and the 2\textsuperscript{nd} level lookup consists of TCAMS in the form of actual pages and the bucket. The concepts of page tables and buckets are described later in this section. The comparator primarily enables page table with entries corresponding to the pages that contain the longest matching prefix. When an IP address is looked up using the forwarding engine, depending on the \textit{cube} that covers the IP address, the comparator enables the appropriate page table entry. For example, the prefixes 128.1.5/24 and 128.1.4/24 can be represented by a single \textit{cube} 128.1.2/23, which is stored in the page table. The 32-bit IP address is again looked up in the page table, and if there is a match all the pages covered by the prefix are enabled. The least significant 32-\textit{w}\textsubscript{1} bits of the IP address are then looked up in the pages to find the longest prefix match. However, if there is no match in the page table the 32 bit IP address is looked up in the bucket for a

![Figure 5: EaseCAM architecture of the proposed approach](image)
match. We now give a detailed description of each of the architectural components.

The page tables store the page ID’s associated with the pages. These page tables are 32-bit wide and implemented using TCAMs. The word length of page tables is 32-bit length because the maximum length of the cube can be as high as 32 bits. The minimum size of the page table is given by $C_{\text{max}}$, as computed in the FillPages algorithm. The page tables have some empty entries (stored with all 0’s, so that they don’t match with any address), which will be used for memory management as discussed in Section IV C.

The range comparator is designed (using TCAM) such that it would selectively enable the TCAM page table that contains the cubes covering the incoming IP address. From the router statistics, we know that the number of page tables required is few in numbers. Thus, the number of comparators required is very small (i.e. bbnplanet statistics showed that the number of comparator varied from 14 to 308 for different values of $w_1$). Hence, the overheads due to comparators for memory usage and power consumption are negligible.

**Bucket**

The proposed scheme stores all the prefixes of length $\leq w_1$ separately into a bucket that is 32 bits wide. These entries will have to be searched only when there is a mismatch in the 1st level. We now define the parameter $\alpha$ as the fraction of routing table entries present in the bucket. Since the bucket contains all the prefixes $\leq w_1$, the value of $\alpha$ increases with the increase in $w_1$. We will see in Section V that the value of total estimated power is mostly controlled by $\alpha$ for larger values of $w_1$ and that is independent of the bucket size for smaller values of $w_1$. 

The statistics from different routers show that the number of prefixes in the bucket is very small for $w_1\leq 16$ (approx 2% of the routing table). Since the growth rate of the prefixes of length $\leq 16$ is small, it is very unlikely that the bucket will overflow when designed with reasonable fill factor. The fill factor for the bucket $\alpha_f$ is defined as the proportion of bucket that are filled with prefixes $\leq w_1$ during router reconfiguration. It may be mentioned here that the bucket is not only used for storing prefixes of length $\leq w_1$, but also prefixes that arrive due to an update which cannot be grouped as a part of a page in the 2nd level. These types of prefixes are rarely encountered and wouldn’t attribute to bucket overflow for small values of $w_1$.

```plaintext
ForEach $w_1$ from 8 to 31
  // These are the possible values of $w_1$
 ForEach $j$ from 1 to 8
    // $\beta$ ranges from 2 to 256, powers of 2
    $\beta = 1 << j$
    $C' = 0$
    Page = FillPage($P$, $w_1$, $\beta$, $Y$, $C'$)
    // Fill the page using heuristics
    Mem = Page * $\beta * (32 - w_1) / 32 + Page + M_c$
    // $M_c$ memory of comparators used
    If (Mem < Mem$_{w_1}$)
      // Obtain optimal $\beta$ value
      Mem$_{w_1}$ = Mem
      Page$_{max}$ = Page
      $\beta_{w_1} = \beta$
  EndFor
EndFor
```

Figure 6: Heuristics for finding optimal page size for all $w_1$

It is important to note that the bucket, page tables and the comparators are all of 32-bit word length, while the pages in the TCAM have word length 32-$w_1$. Thus we cannot place them in the same TCAM chip as the prefixes. However, the total memory
for the buckets, page tables and comparators will be of very small size, and they can be implemented using smaller sized TCAM chips of word length 32 bits each.

**Optimal page size**

We now present a heuristics to find the optimal page size for a given $w_1$ so that the total memory requirement is minimal. The algorithm is described in Figure 6, that computes the optimal value of $\beta = \beta_{w_1}$ for each value of $w_1$. The sum total of $\text{Mem}_{w_1}$ and the bucket size will give the total memory required for $w_1$ bits in the 1st level.

**D. Empirical Model for Memory**

We now present empirical models to compute the total memory requirements, energy consumption and access time for the proposed architecture. We use $\alpha$ as the maximum bucket size, $\beta$ as the page size, $\gamma$ as the fill factor of the pages and $N$ as the number of entries in routing table after compaction. Also from the algorithm we see that $\beta_{w_1}$ represents the optimal page size for $\text{Page}_{\text{max}}$ pages that minimizes memory for $w_1$ bits in the 1st level lookup. The minimum memory requirement due to the proposed architecture (32 bit entries equivalent) is:

$$= \beta_{w_1} \cdot \text{Page}_{\text{max}} \cdot (32-w_1)/32 + \text{Page}_{\text{max}} + \text{Page}_{\text{max}}/C_{\text{max}} + N\alpha/\alpha_f$$ …(1)

We have approximated the memory used by the page tables to the number of pages used for optimal $\beta_{w_1}$. This is a reasonable approximation since the range comparators can be designed to squeeze the entire page ID’s into the page tables with a negligible wastage in memory. The number of comparators used can be approximated to the number of page tables and is given by $C_{\text{max}}$. The bucket size is represented by $N\alpha/\alpha_f$, and $N/\alpha$ represents the number of prefixes in the bucket during reconfiguration time. We believe
that the prefixes that will be added in the future are very few since those are rarely occurring prefixes. This observation is valid for small values of \(w_1\).

IV. INCREMENTAL UPDATES

The dynamically changing routing table could have 100s of updates per second. Though most of the updates (inserts/deletes) are route flaps, still 10s of updates per second will be required in backbone routers. Thus we need a fast incremental update algorithm for the proposed architecture.

A. Insertion

The architectural design proposed in this paper is very conducive to the fast incremental updates. When a new prefix \(P_i\) is to be added into the TCAM, we first check if it is one of the rarely occurring prefixes. This could be done by checking if \(|P_i| \leq w_1\) or if the prefix is not covered by any cube \(C_i\) that is present in the page table, and added to the bucket. However, if the prefix is covered by some cube \(C_i\), there could be a maximum of 3 cubes that covers this prefix \(P_i\). Based on the heuristics on storing the prefixes into TCAM pages, we know that the prefix must be inserted into the page whose page ID \((C_i)\) satisfies the relation: \(|LCS(Cover(C_i))|=|LCS(P_i)|\). Before we insert the prefix \(P_i\) into a cube \(C_i\), we minimize the new prefix with all the P prefixes that are covered by \(C_i\) \((Cover(C_i))\), and have the same LCS as the new prefix \(P_i\). Then the minimized set of prefixes is updated into the TCAM page with page ID \(C_i\). We need to update the value of \(C_i\) based on the state of the current prefixes in the page. This is done by the \(UpdatePageTable(C_i)\) procedure in the insertion heuristic given in Figure 7.

B. Deletion

The deletion process is similar to the insertion process but is simplified by the fact that the minimization is done on the raw prefix data and not on the previously minimized set.
Since the number of such prefixes cannot exceed 256, it is not an overhead in terms of computation.

It is important to note that the $\text{Cover}(C_i)$ function will return the set of prefixes that have the same LCS as the new entry, but without being minimized. As we have shown earlier in Observation II.1, this will not result in minimizing more than 256 entries. The details of $\text{InsertToTCAM}$, $\text{InsertToBucket}$, $\text{DeleteFromTCAM}$, $\text{DeleteFromBucket}$ and $\text{UpdatePageTable}$ schemes are not provided due here to lack of space.

```plaintext
Deletion(Pi)
If( \mid Pi \mid < w1)
DeleteFromBucket(Pi)
Else
If(\exists Ci that covers Pi)
   Find Ci such that
   \mid LCS(P\text{Cover}(C_i))\mid = \mid LCS(P_i)\mid
   P=\text{Cover}(Ci)
   P=P-Pi
   Q=\text{Minimize}(P)
   DeleteFromTCAM(Q' \cap P)
   InsertToTCAM(Q \cap P')
   UpdatePageTable(C_i)
Else
   DeleteFromBucket(P_i)
```

Figure 8: Incremental deletion

The runtime for the update algorithms (insertion and deletion) are bounded by the time due to minimization using Espresso-II algorithm. Table II gives the runtimes to minimize the prefixes using the proposed approach and one due to the technique used in [9]. The size index in Table II represents the maximum input that would to be given to the Espresso-II algorithm for minimization during a prefix update. The Espresso-II algorithm was run on an Athlon dual processor running 1.6 GHz, to minimize the
prefixes\(^1\) when a request for updating a new prefix is issued by the neighboring router using the approach in [9], the worst-case incremental prefix update took 63.04 sec for the *bbnplanet* router and 1098.47 sec for *attcanada* router, which appear to be expensive for high-performance routers. The proposed approach on the other hand only takes 0.006sec for incremental prefix update. The value is not only small and practical, but also bounded since at any point of time the number of inputs to Espresso-II algorithm never exceeds 256. The proposed algorithm uses 8 bits as opposed to 32 bits in [9] and hence makes the computation fast.

**TABLE II: Comparison of incremental update time**

<table>
<thead>
<tr>
<th>Router</th>
<th>Total Prefixes</th>
<th>Approach in [9]</th>
<th>Proposed Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size</td>
<td>Time (sec)</td>
<td>Size</td>
</tr>
<tr>
<td><em>attcanada</em></td>
<td>112412</td>
<td>15146</td>
<td>1098.47</td>
</tr>
<tr>
<td><em>bbnplanet</em></td>
<td>124538</td>
<td>7580</td>
<td>63.04</td>
</tr>
</tbody>
</table>

**C. Memory Management**

When the router is reconfigured the comparators, page tables and pages are setup based on the optimal architecture for the router. The design of the router is based on the prefix statistics and hence it is less likely that the pages will overflow. Also, the page fill factor \(\gamma\) ensures that during the build time of the router the pages have enough space for future updates. Also, most of the prefix updates are route flaps. So it is more likely that the same set of prefixes will be added and deleted that would result in very less chance of an overflow. However, it is still possible that the pages could overflow. In the previous

---

\(^1\) Prefixes here indicate the routing table prefixes after initial compaction (prefix overlapping and minimization)
section when we discussed the insertion algorithm we did not consider the possibility of an overflow. Hence we present a memory management technique that will effectively reorganize the pages without affecting the update time. Let \( P_i \) be the prefix that resulted in the overflow.

Let \( Q_{w1} = p_{i1}p_{i2}p_{i3}..p_{iw1} \)

For all \( P_i \) such that \( Q_{w1} \) covers \( P_i \)

- FillPages(\( P_i, w_1, \beta_{max}, 1, C_i \))
- UpdatePageTable(\( C_i \))
- ReprogrammeDeMux(\( C_i \))

Figure 9: Memory management heuristic

The memory management algorithm ensures that only the pages that have their Page ID same as \( C_i \) are recomputed during an overflow. It is important to note that if an overflow results in creating an extra page, the free pages already present in the TCAM will be used and the page tables will be updated appropriately. The memory management technique assumes that the page table will not overflow. Though this is very unlikely to happen, it may not be ruled out. In such a case we reconfigure the router all over again.

The deletion and insertion of prefixes could also result in page fragmentation. To overcome this problem, all the pages that have the same page ID have prefixes sorted in the increasing order of prefix length. We then adopt the technique discussed in [7] to efficiently insert and remove entries within the TCAM to reduce fragmentation.

V. RESULTS AND CASE STUDIES

In this section we present the results based on the proposed architecture. The results have been illustrated with two real-life large routers (bbnplanet and attcanada) to demonstrate the effectiveness of the proposed approach. We compute the value of \( w_1 \)
for which power is bounded and memory requirement is the least. Finally, we evaluate the results corresponding to above two router examples.

A. Power Consumed per Lookup

The power consumption in TCAM architecture is determined in terms of number of active TCAM entries that are enabled during the lookup operation. The number of active entries in architecture depends on the selection of parameter $w_1$. Based on this we explore the range of values of $w_1$ for which power is bounded. We know that the power consumption is also based on number of entries looked up in comparators, page table and the bucket. As discussed earlier, the number of entries in the comparator and page table is constant and negligible for a particular value of $w_1$. However, the number of active pages and buckets during the lookup operation depends on the match/mismatch in the 1st level comparator TCAM. Thus the bound on maximum power consumption per lookup depends on the size of the bucket and the number of active pages looked up in the 2nd level TCAM. From observation III.1 we know that the number of entries looked up in the pages is a constant.

![Figure 10 (a): Power consumption in bbnplanet router](image1)

![Figure 10 (b): Power consumption in attcanada router](image2)
In order to find all the values of $w_1$ for which power is bounded we examine all values of $w_1$ for which the bucket size is $\leq$ page size. For those values of $w_1$, we can confirm that the power is bounded by the number of pages enabled in 2nd level (256*3). We have used $\alpha_t$ as the fill-factor to bucket design and considered its value as 0.5 in our simulation to ensure that 50% of the bucket space is available for future updates. We have also used a fill factor ($\gamma$) of 0.5 for the pages in our page filling heuristics.

We plot the number of active entries in the bucket during lookup operation for varying $w_1$ and keeping the page entries to its bound. Figure 10a and 10b show the results for bbnplanet and attcanada routers respectively. It is evident from these figures that for $w_1 \leq 13$, we obtain a bound on the power consumption for both the routers. It may be worthwhile to mention that we have not shown the values for $w_1 > 16$, since for such values of $w_1$ the power consumption per lookup can be very high and will not be useful to router design.

C. Memory Requirements

The total memory requirements at different values of $w_1$ are given by the empirical equation in Section III.D. We use this equation to find the optimal value of $w_1$. 

![Total Minimum Memory Requirements](image)
for which power is bounded and memory requirement is the least. From the above
discussion, we observe that in the cases of bbnplanet and attcanada, power is
bounded for $8 \leq w_1 \leq 13$ (Figure 10a and 10b). We use this information to find the optimal
value of $w_1$ for which memory requirement is the least. From the plot given in Figure 11,
we see that the values of $w_1$ for which the memory requirement is the least and that
satisfy the power requirement are 13 and 12 for bbnplanet and attcanada routers
respectively.

It is important to mention that we have not considered the value of $w_1 < 8$ in this
paper as we observed that there is an increase in memory consumption when $w_1$ is
decreased from 8 through 1, though the power requirement is bounded by the 256*3 page
entries.

D. Case Studies

Tables III & IV show the reduced memory requirement and access time for both the
routers as we apply compaction and architectural technique successively without fill-
factors ($\gamma=1$ and $\omega=1$).

**TABLE III. Reduction in memory requirements**

<table>
<thead>
<tr>
<th>Router</th>
<th>Raw data (entries)</th>
<th>After Compaction (entries)</th>
<th>Effect of Architecture (entries)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attcanada</td>
<td>112412</td>
<td>57837</td>
<td>50182</td>
</tr>
<tr>
<td>Bbnplanet</td>
<td>124538</td>
<td>71500</td>
<td>59883</td>
</tr>
</tbody>
</table>

We see that about 40% compaction is obtained in the case of bbnplanet router with
access time reduced by about 55%.
The Table IV shows the power reduction as an equivalent to the number of TCAM entries. Column 3 in Table IV indicates the reduction in number of entries looked up due to the compaction and column 4 is due to the use of architectural techniques. The power estimation is based on the number of entries enabled during a lookup process. The large routers like att.canada and bbn.planet can be designed with low power per lookup using the EaseCAM approach.

<table>
<thead>
<tr>
<th>Router</th>
<th>Raw data (entries)</th>
<th>After Compaction (entries)</th>
<th>Effect of Architecture (entries)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attcanada</td>
<td>112412</td>
<td>15146</td>
<td>669</td>
</tr>
<tr>
<td>Bbnplanet</td>
<td>124538</td>
<td>7580</td>
<td>768</td>
</tr>
</tbody>
</table>

VI. RELATED WORK

In [3], authors discussed the use of TCAM in hardware search engine and called that as Database Accelerator. The fast routing table lookup using TCAM was proposed in [5] wherein the authors used sorting of prefixes to table lookup. Subsequently, Shah and Gupta [7] provided the techniques that avoid the overhead of sorting. Kobayashi et al. associated priority to each TCAM entry in [2] and showed the use of extra hardware for output for multiple prefix matches. Their approach avoids sorting requirements at the cost of increased latency due to the extra hardware. The authors in [1] have tackled the IP lookup problem using specialized hardware. Authors in [8] and [11] have proposed the compaction technique based on prefix pruning and showed significant prefix compaction. In [12] authors have suggested an on chip minimization algorithm for compacting prefixes. In [9], the authors proposed to place TCAMs on separate bus for parallel access
and introduced paged-TCAM architecture to increase the throughput and reduce the power consumption in TCAM routers. Recently, the authors in [10] proposed bit-selection architecture and partitioning technique to design power efficient TCAM architecture.

VII. CONCLUSION

In this paper we presented a novel architecture for a TCAM-based IP forwarding engine. We have shown significant reduction in memory usage based on the prefix compaction and architectural design. We designed heuristic to store entries in TCAM pages so that only a bounded number of entries are looked up during the search operation. A fast incremental update algorithm has been introduced that is time bounded. We also proposed efficient memory management technique to tackle the overflow problem. The memory requirements, power consumption and delays for router architecture have been outlined. To demonstrate the merit of the proposed architecture, we used the architectural features on bbnplanet and attcanada routers based on their trace statistics and evaluated the benefits of our approach. It has been shown that the memory requirement is reasonably low due to use of effective compaction technique. At the same time the power consumption is found to be remarkably low to promise efficient TCAM design in the future.

VII. REFERENCES


