An Ultra High Throughput and Power Efficient TCAM-Based IP Lookup Engine[i]

Kai Zheng, Chengchen Hu, Hongbin Lu, Bin Liu
Department of Computer Science and Technology
Tsinghua University
Beijing, China 100084
Zk01@mails.tsinghua.edu.cn, Chengchenhu@yahoo.com.cn,
Lu-hb02@mails.tsinghua.edu.cn, Liub@tsinghua.edu.cn

***Abstract*--Ternary content-addressable memory (TCAM) is widely used in high-speed route lookup engines. However, restricted by the memory access speed, the route lookup engines for next-generation terabit routers demand exploiting parallelism among multiple TCAMs. Traditional parallel methods always incur excessive redundancy and high power consumption. We propose in this paper an original TCAM-based IP lookup scheme that achieves an ultra high lookup throughput and a high utilization of the memory while being power efficient. In our multi-chip scheme, we devise a load-balanced TCAM table construction algorithm together with an adaptive load balancing mechanism. The power efficiency is well controlled by decreasing the number of TCAM entries triggered in each lookup operation. Using 133MHz TCAM chips and given 25% more TCAM entries than the original route table, the proposed scheme achieves a lookup throughput of up to 533Mpps and is simple for ASIC implementation.**

***Keywords*—TCAM, route lookup, throughput, power consumption**

I. Introduction

IP address lookup is one of the key issues in designing high performance routers. The challenge arises from that a) the length of IP prefix is variable and the incoming packet does not carry the prefix length information for IP lookup; b) one IP address may match multiple prefixes in the forwarding table and the longest matching prefix, or *longest matching prefix* (LMP), should be chosen; c) advances in fiber-optic technology is pushing the line rate of core routers to 40Gbps or even higher, which implies that a single line card would support the packet processing rate of 128Mpps (packets per second) or higher.

Currently, packet forwarding based on CIDR IP addresses is well understood with both trie-based algorithms and TCAM-based schemes. Although many alterations have been proposed to optimize the original trie structures [1-3], the lookup speed in trie-based mechanisms using DRAMs or SRAMs can hardly be further improved because of its intrinsic characteristic. *Ternary content addressable memory* (TCAM) is a fully associative memory that allows a “don’t care” state to be stored in each memory cell in addition to 0s and 1s. It is a promising device to build a high-speed LMP lookup engine, because it returns the matching result within a single memory access time. In contrast, many trie-based algorithms may require multiple memory accesses for each lookup. Moreover, updating the forwarding table in TCAM-based schemes is generally simpler than that in trie-based algorithms.

However, the high cost to density ratio and low power efficiency of the TCAM are traditionally the major concerns in building the lookup engine. Furthermore, although the TCAM can finish each lookup with a single memory access, the lookup throughput is restricted by the TCAM access speed. Current TCAM can work at a speed up to 266MHz [4], which still cannot meet the need of next-generation terabit routers. Recent developments in TCAM technology have effectively addressed the issue of high cost to density ratio. There are commercially available TCAM devices with the density up to 18M bits per chip and the costs that are comparable with the trie-based schemes [4-6].

Many researches have been conducted to optimize the TCAM-based lookup engines [7-11]. Liu *et al.* [7] use both pruning techniques and logic minimization algorithms to reduce the size of TCAM-based forwarding tables, which in turn reduces the cost and power consumption of the TCAMs. However, the power consumption is still quite high. Zane *et al.* [8] developed a power efficient lookup engine benefited from a new feature of some TCAMs called “partition-disable”. The idea is that during a parallel lookup operation, not all entries in the table are compared. If a large forwarding table can be partitioned into small partitions and only the one containing the prefixes that match the incoming IP address is enabled during the lookup operation, the power consumption of the TCAM can be dramatically reduced. However, it requires the hardware support of the new feature.

We presented a simple distributed parallel lookup algorithm in [12]. Using 4-8 memory chips for parallel lookup operations and assuming that the traffic is evenly

[i] This research is supported by NSFC (No.60173009 & No.60373007) and National 863 High-tech Plan (No.2003AA115110).

distributed among IP prefixes, the proposed scheme achieves a lookup speedup of 3-7, while the memory requirement remains unchanged. Nevertheless, due to the facts that the traffic load among the IP prefixes is not so evenly distributed, the lookup throughput becomes non-deterministic. Panigrahy *et al.* proposed a multi-chip mechanism to improve lookup throughput [10]. In their scheme, eight or more TCAM chips are used for parallel lookup operation. Based on certain assumptions of the traffic distribution and by doubling the number of entries (duplicating the TCAM contents), a speedup factor of five is achieved. However, eight chips need eight separate data buses (a large pin count) and the traffic distribution ASIC needs to work at 8*133MHz>1GHz, which is difficult to implement. The doubled number of route entries also distinctively increases of the cost and power consumption, which makes it non-scalable.

In this paper, we propose an ultra high throughput and power efficient IP lookup scheme using regular low cost TCAMs to satisfy the demand of next-generation terabit routers. Our scheme employs multiple memory modules for parallel table lookup operations, which is the key to break the restriction of the TCAM access speed and to reduce the power consumption. We have also devised the algorithms to solve the two main issues in applying chip-level parallelism. The first issue is how to evenly allocate the route entries among the TCAM chips to ensure a high utilization of the memory. The second one is how to balance the lookup traffic load among the TCAMs to maximize the lookup throughput.

The rest of the paper is organized as follow. Section II presents a distributed TCAM table organization and the load-balance-based table construction algorithm. Section III describes the complete architecture of the proposed ultra high throughput and power efficient IP lookup engine. Section IV presents the theoretical performance estimation based on queuing theory and the simulation results of the proposed scheme. Section V gives the concluding remarks.

II. Distributed Parallel TCAM Organization and Load-Balance-Based Table Construction Algorithm

Our scheme exploits the parallelism among multiple TCAMs to increase the lookup throughput. By the analysis of the existing route tables, we first introduce a method to divide the forwarding table into small segments. A assume that it is known here. *D_prefix*[*i*] is defined as the ratio of the traffic load of *prefix*[*i*] to the total bandwidth. Therefore,

$$\sum_i D\_prefix[i] = 1. \qquad (2.1)$$

Definition 7:

We introduce a concept of storing *redundancy rate* here. The redundancy rate of a specific prefix (or ID group) is defined as the number of duplications of this prefix (or ID group) stored in the TCAM chips. The redundancy rate of the entire scheme is *M/N*.

Theorem 1:

In the (extended) forwarding table, two prefixes with different IDs do not need to be stored in any specific order when applying the longest prefix matching operation. As long as the prefixes in an ID group are stored in the decreasing order of prefix lengths, the TCAM(s) will return the correct matching result.

Proof:

TCAM-based matching requires that *Prefix 1* should be stored in lower address than *Prefix 2* if *Prefix 2* is the prefix of *Prefix 1;* but two prefixes each one of which is not a prefix of the other can be stored in arbitrary order. Since each prefix in the extended forwarding table has a length of 13 or more, the first 13 bits of two prefixes with different IDs must be different to each other. Therefore, they need not to be stored in any specific order.

B. Analysis of the Existent Route Table

We have analyzed the route table snapshot data provided by the IPMA[iv] project and found out that the route prefixes can be evenly classified into groups by certain bits of themselves. We use the (10-13)-th bits of the prefixes (their **ID**s) in the four route tables to classify them into 2^4=16 groups, and get the result as shown in Fig. 2. All of the four tables show quite even distribution. It also tends to be that the larger the route table is, the better result we can get.

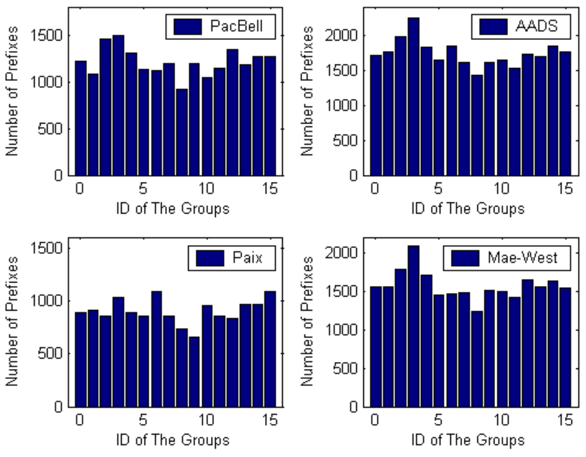

Figure 2. Prefixes distribution among ID groups in four real-life route tables.

[iv] A joint effort of the University of Michigan and Merit Network, on http://www.merit.edu/ipma.

C. Distributed Memory (TCAM) Organization

We have introduced a simple but efficient method to partition the forwarding table into small segments by using the ID bits of the prefixes. Since multiple TCAMs are used in our scheme, the next step is to allocate these ID segments (groups) to each TCAM. Based on the above route table analysis, we know that the prefixes are evenly distributed into each ID group the size of which is approximately *N*/16. Therefore, we may partition each TCAM into small blocks with a size of *N*/16 (or slightly larger) so that each block can store one ID group. Now we use the ID group as the basic unit to allocate the prefixes among the TCAMs. As Theorem 1 dictates, as long as the prefixes in an ID group are arranged in the decreasing order of prefix length, the TCAM will return the LMP as the result. Fig. 3 shows an example of the TCAM organization.

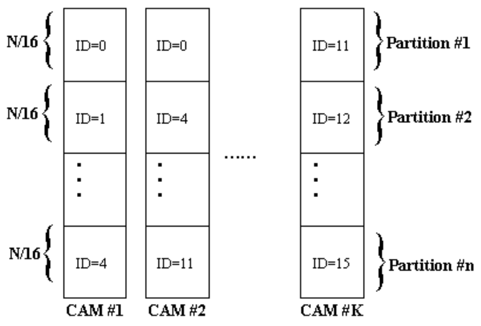

Figure 3. An example of the TCAM organization.

D. Load-Balance-Based Table Construction Algorithm

Allocating the prefixes evenly and balancing the lookup traffic among the TCAMs are the two tasks of the proposed table construction algorithm. We have addressed the first task in the subsection above. For the second task, we first calculate the load distribution of the ID groups, $D\_id[j], (j = 1...16)$, by summing up the distribution of the prefixes in the same ID group:

$$D\_id[j] := \sum_{\text{prefix}[k] \in j} D\_prefix[k]. \qquad (2.2)$$

Two methods can be then introduced to balance the lookup traffic among the TCAMs. Firstly, we can use $D\_id[j](j = 1...16)$ as the weights of the ID groups and make the sum of the weights of the ID groups in each TCAM balanced. Secondly, for the ID groups with large weights, we may introduce storing redundancy into the scheme. By duplicating the prefixes in a specific ID group to multiple TCAMs, we distribute the traffic of the ID group among these TCAMs. For example, if ID group *j* is allocated (duplicated) to three TCAMs, then each one of these three chips bears a traffic load of *D_id[j]*/3. Our proposed algorithm is the combination of these two methods. Before we describe the table construction algorithm, a mathematical model of the problem is introduced.

Mathematical Model of the Problem:

Let *K* be the number of TCAM chips; *Rd* be the storing redundancy rate of the whole mechanism; *T* be tolerance of the discrepancy of table sizes among the TCAM chips; *P* be the length of the ID segment (It is assumed to be 4 in the proposed mechanism) and there should be 2^P ID groups. *K, Rd, T,* and *P* are given.

Let *S* be the set of all the ID groups, $S=\{1,2,...,2^P\}$; Q_k $(k=1,...,K)$ be the set of the ID groups that are allocated to TCAM chip #*k*, $\forall Q_k$, $Q_k \subseteq S$ and $\bigcup Q_k = S$ $(k=1,...,K)$. And $|Q_k|$ denotes the number of elements (ID groups) that Q_k contains.

We define the number of TCAMs that ID Group #*j* is allocated into as *G[j]* $(j \in S)$; *G[j]* denotes the storing redundancy rate of the ID group #*j*.

We define *W[j]* as the incremental traffic load distributed to a TCAM chip when ID group *j* is allocated to this chip, so $W[j] := D\_id[j]/G[j], (j \in S)$, and we call *W[j]* Partition-load of ID group #*j*;

We define *D[k]* as the traffic load allocated to TCAM chip #k , so $D[k] := \sum_{j \in Qk} W[j]$ $(k=1,...,K)$; then the optimization problem is given by

The Load-Balance-Based Table Construction Problem

Subject To.

$$Q_j \subseteq S, j=1,2,...,K\,,\ \bigcup_j Q_j = S\,; \qquad (2.3)$$

$$\big||Q_i|-|Q_j|\big| \le T \quad (0<i\le K, 0<j\le K)\,; \qquad (2.4)$$

$$BOOL(i,j) := \begin{cases} 1.........j \in Q_i\,; \\ 0........j \notin Q_i\,; \end{cases} \qquad (2.5)$$

$$G[j] := \sum_{i=1}^{k} BOOL(i,j) \quad (j \in S); \qquad (2.6)$$

$$\sum_{j \in S} G[j] \le 2^P \times Rd \qquad (j \in S); \qquad (2.7)$$

$$1 \le G[j] \le K\,,\ \ G[j] \in Z^+ \quad (j \in S); \qquad (2.8)$$

$$D[k] := \sum_{j \in Qk} D\_id[j]/G[j] \quad (k = 1,..., K)\,; \qquad (2.9)$$

Minimize: $\underset{k}{Max}(D[k]) - \underset{k}{Min}(D[k]) \quad (k = 1,..., K)$.

This problem is proven to be NP-hard (please see Appendix for the proof). Therefore, we give a greedy algorithm to solve the problem.

The task of this algorithm is to figure out an allocation scheme ({Q_k}, *k = 1,...,K*) of the ID groups, with which the lookup traffic distribution among the TCAM chips (*D[k], k= 1,...,K*) is as even as possible.

Based on the traffic distribution (*D_id[j],* $j \in S$), the redundancy rates of the ID groups (*G[j],* $j \in S$) are calculated iteratively aiming at maximizing the sum, so as to maximize the TCAM space utilization (while satisfying restriction 2.7). Then the ID groups are allocated to TCAM chips following two principles: 1) ID groups with larger *Partition-load* (*W[j]*) are allocated more preferentially; 2) TCAM chips with lower load are allocated to more preferentially.

Load-Balance-Based Table Construction Algorithm

```
1) Pre-Calculation: /* Calculate $G[j]$ iteratively*/
    Initialize $a=0, b=1, \lambda=(a+b)/2$ ,
               $Obj = \lfloor 2^P \times Rd \rfloor$ [v],
               $precision = 0.001$ .
    We define function $F(X)$ as
    {
      $P'[j] = 2^P \times Rd \times X \times D\_id[j] \quad (j \in S)$;
      $P[j] = Min\{\lceil P'[j] \rceil, K\}$ [vi] $\quad (j \in S)$;
      $Y = \sum_{j=1}^{k} P[j] - Obj$;
       return $Y$;
    }
    /*As it is obvious that $F(0)<0$, $F(1) \geq 0$, and F(x) is a
    monotonic function, we can find a solution to $F(\lambda)=0$,
    $\lambda \in (0,1]$ */
    while $F(\lambda) \neq 0$ do
      if $F(\lambda) < 0$ then
        $a = \lambda$;  /*a denotes the lower bound*/
      else
        $b = \lambda$;  /*b denotes the upper bound*/
      end;
      $\lambda = (a+b)/2$;
      if $|b-a| \le precision$ then
        break;
      end if;
    end while;
    $G'[j] = 2^P \times Rd \times \lambda \times D\_id[j] \quad (j \in S)$;
    $G[j] = Min(\lceil G'[j] \rceil, K) \quad (j \in S)$;
    Let $W[j] = D\_id[j]/G[j] \quad (j \in S)$;
2) Initialization:
      $Q_k = \Phi \quad (k=1,...,K)$;
      $D[k] = 0 \quad (k=1,...,K)$;
      $i = j = k = 0$;
3) Sort $\{i,\ i=1,2,...,2^P\}$ in the decreasing order of $W[i]$, and
record the result as $\{Sid[1], Sid[2],..., Sid[2^P]\}$;
4) for $i$ from 1 to $2^P$ do
      for $j$ from 1 to $G[Sid[i]]$ do
        Sort $\{k, k=1,...,K\}$ in the increasing order of
         $D[k]$, and record as $\{Sc[1], Sc[2],..., Sc[K]\}$;
        for k from 1 to $K$ do
          if $Sid[i] \notin Q_{Sc[k]}$ and $|Q_{Sc[k]}| < \underset{i=1...K}{Min}|Q_i| + T$
           then $Q_{Sc[k]} = Q_{Sc[k]} \cup \{Sid[i]\}$;
```

---

[v] $\lfloor\ \rfloor$ Denotes a Round operator here.
[vi] $\lceil\ \rceil$ Denotes a Ceiling operator here.

```
                $D[Sc[k]] = D[Sc[k]] + W[Sid[i]]$;
                **break**;
            **end if**;
        **end for**;
    **end for**;
**end for**;
```

5) output $\{Q_k, k = 1,\dots, K\}$.

According to our simulation results, the iteration in step 1 converges quickly. By applying the greedy algorithm, fairly good result can be achieved when all of the restrictions are satisfied.

An Example:

Suppose that the traffic distribution among ID groups is given by Table I.

TABLE I. TRAFFIC DISTRIBUTION AMONG ID GROUPS

ID	***0***	***1***	***2***	***3***	***4***	***5***	***6***	***7***
D_id%	1	2	3	5	19	7	8	8
ID	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***
D_id%	6	3	7	2	19	2	2	6

If Rd=1.25, T=1, and K=4, then the allocation result given by the algorithm is shown as Table II.

TABLE II. PREFIXES ALLOCATION (IN ID GROUPS)

Group ID		**Partition #**				
		I	***II***	***III***	***IV***	***V***
CHIP#	***#1***	10	**4**	**12**	2	13
	#2	5	**4**	**12**	3	0
	#3	7	**12**	15	1	11
	#4	6	**4**	8	9	14

And the traffic load among the 4 chips is given by Table III.

TABLE III. TRAFFIC LOAD AMONG THE CHIPS

Chip #	***1***	***2***	***3***	***4***
Load%	24.67	25.67	24.33	25.33

E. Analysis of the Power Efficiency

Traditional TCAM is a fully parallel device. When the search key is presented at the input, all entries are triggered to perform the matching operations, which is the reason of its high power consumption. Current high-density TCAM devices consume as much as 12–15Watts per chip when all the entries are enabled for search. In fact, however, the power consumption of the TCAM can be lowered down. We found that not all the entries need to be triggered simultaneously during a lookup operation. The entries that really need to be triggered are the ones matching the input key. Thus, the way of making the power consumption efficient is to avoid the non-essential entries from being triggered during a lookup operation. In order to measure the efficiency of power consumption in each lookup operation, we introduce the concept of *dynamic power efficiency* (*DPE*), as defined below:

Definition: Dynamic Power Efficiency (*DPE*)

Let V_i be the number of entries triggered during a specific lookup operation and matching the input search key i; W be the total number of entries triggered during a specific lookup operation.

We define $DPE(i)$ as:

$$DPE(i) := V_i / W; \qquad (2.10)$$

For instance, suppose that the input search key i is "A.B.C.D", and there are five route prefixes matching this key in the route table, which are: A/8, A.B/12, A.B/16, A.B.C/24, and A.B.C.D/32. And the size of the total route table (all in a single chip) is 128,000, and all the entries are triggered during a lookup operation, then

$$DPE(i) = 5/128{,}000 = 3.90625\times 10^{-5}.$$

There are two ways to decrease the number of entries triggered during a lookup operation. One is to store the entries in multiple small chips instead of a single large one. The other is to partition one TCAM into small blocks and trigger only one of them during a lookup operation. If the hardware (TCAM) supports the partition-disable function, the proposed scheme can be benefited from both methods.

We still use the example mentioned above. If we use four small TCAMs instead of (to replace) the large one and partition each of them into 8 blocks to store the route prefixes, assuming that the storing redundancy rate is 1.25, then the number of prefixes that stored in each of the $4\times 8 = 32$ blocks should be $128{,}000\times 1.25/32 = 5{,}000$. Applying the proposed TCAM partition method and supposing that the hardware supports the partition-disable feature, the *DPE* is now increased to $5/5{,}000 = 1.0\times 10^{-3}$, meaning that the power consumption efficiency is increased by a factor of more than 25.

III. COMPLETE IMPLEMENTATION ARCHITECTURE

The complete implementation architecture of the ultra high throughput and power efficient lookup engine is shown in Fig. 4. Given an incoming IP address to be searched, the ID bits of the IP address are extracted and delivered to the Indexing Logic for a matching operation. The Indexing Logic will return a set of partition numbers indicating which TCAMs and which block inside each TCAM may contain the prefixes matching the IP address. The priority selecting logic (adaptive load balancing logic) selects one TCAM with the shortest input queue (FIFO) from those containing the matching prefixes and sends the IP address to the input queue corresponding to the selected TCAM. In order to keep the sequence of the incoming IP addresses, a tag (the sequence number) is attached to the IP address being processed.

A. The Indexing Logic

The function of the Indexing Logic is to find out the partitions in the TCAMs that contain the group of prefixes matching the incoming IP addresses. Fig. 5 shows the structure of the Indexing Logic. It is composed of groups of parallel comparing logics. Each group corresponds to one TCAM and each comparing logic corresponds to a partition in the TCAM. The *Index* field represents the ID of the prefix group and the *Partition* field represents the block in which this group of prefixes is stored. Each group of the comparing logic has a returning port. If the ID bits of the incoming IP address match one of the indexes in a group, the corresponding partition number is returned; otherwise, a partition number "111" (7) will be returned, which represents the no-matching information. Because the data width of the comparing logic is short and fixed and only simple "compare" operation is required, it can run at a very high speed.

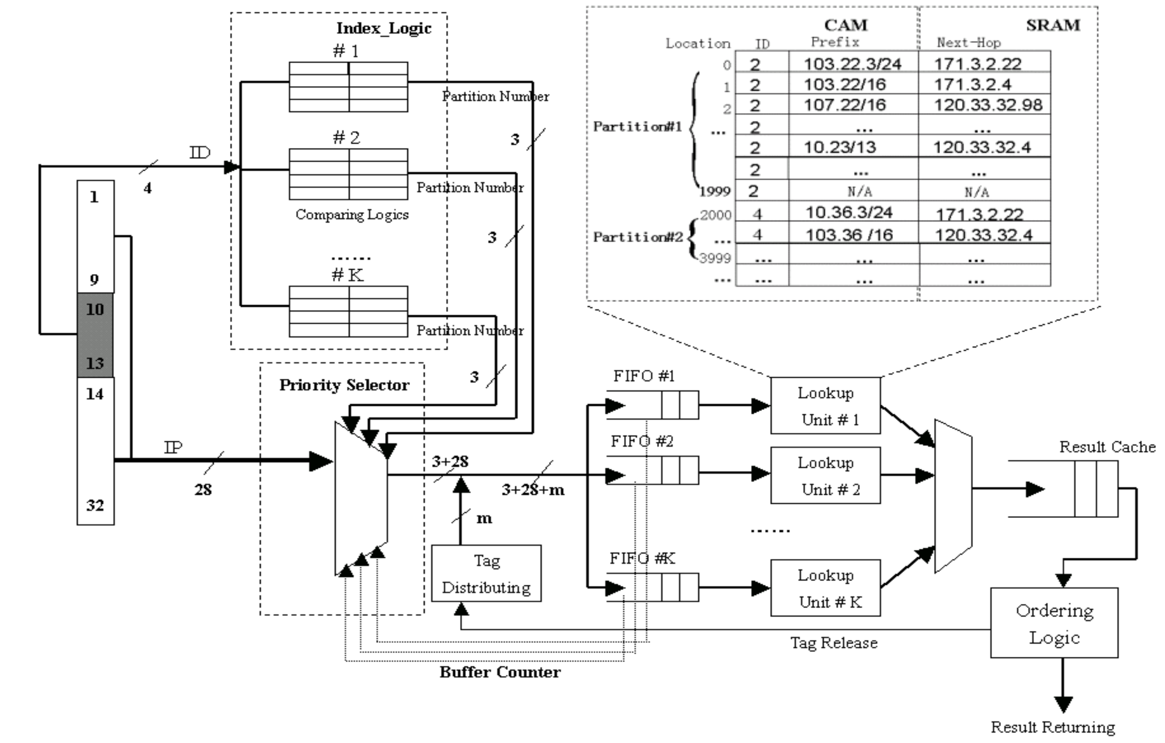

Figure 4. Schematics of the complete implementation architecture.

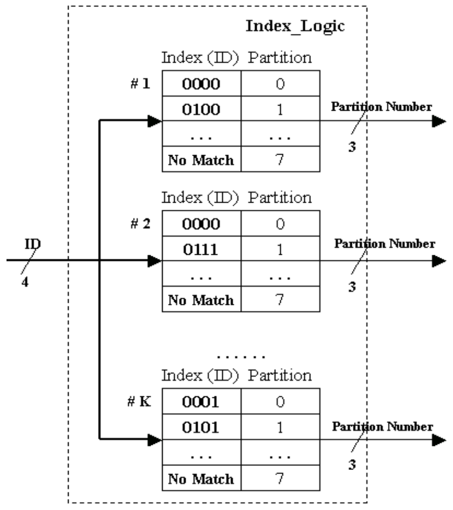

Figure 5. Schematics of the indexing logic.

B. The Priority Selecting Logic (Adaptive Load Balancing Logic)

The function of the Priority Selector is to allocate the incoming IP address to the "idlest" TCAM that contains the prefixes matching this IP address, so that the lookup traffic is balanced among the TCAMs adaptively. As mentioned before, we introduce some storing redundancy into our scheme to guarantee the lookup throughput and one prefix may be stored in multiple TCAMs based on the traffic distribution among the ID groups. The priority selector uses the counter's status of the input queue for each TCAM to determine which one the current IP address should be delivered to. We give the algorithm of the Priority Selector as follow.

Algorithm for the Priority Selector

```
------------------------------------------------------------------------
Input: Counter[i], $i = 1,...,K$ : Status of the buffer for each
          TCAM;
          PN[i], $i = 1,...,K$ : Partition numbers of the chips,
          which are inputted from the Indexing Logic;
Output: Obj:  Serial number of the chip that the current IP
          address should be delivered to.
for i from 1 to K do    /* To find a feasible solution */
    if $PN[i] \neq 7$
    then Obj=i ; break;
    end if;
end for;
for i from Obj+1 to K do
    if (Counter[i]< Counter[Obj])   and   ($PN[i] \neq 7$)
    then
        Obj=i ;
    end if;
end for;
------------------------------------------------------------------------
```

Because the status of queue counters are independent of the current IP address, and no very high precision is required here, this component can also be implemented in high speed ASIC easily.

C. The Ordering Logic

Because multiple input queues exist in the proposed scheme, the incoming IP address will leave the result cache (as shown in Fig. 4) in a different sequence from the original one. The function of the ordering logic is to insure that the results will be returned in the same order as that of the input side. An architecture based on Tag-attaching is used in the ordering logic. When an incoming IP address is distributed to the proper TCAM, a tag (sequence number) will be attached to it. Then, at the output side, the ordering logic uses the tags to reorder the returning sequence.

D. An Example

Assuming that the traffic distribution among the ID groups is given as shown in Table I, we use the proposed load-balance-based algorithm to construct the tables in the TCAMs and get the result as shown in Table II. When an IP address, 166.103.142.195, is presented to the lookup engine, its ID "1100" (12) is extracted and sent to the Indexing Logic. The ID is compared with the 20 indexes in four groups simultaneously. The partition numbers are returned as "010" (2), "010" (2), "001" (1), and "111" (7), which means that TCAM #1, #2, and #3 contain the group of prefixes matching the IP address, while TCAM #4 does not. These results are then sent to the Priority Selector. Suppose that the counter values of the three TCAMs (#1, #2, and #3) are 6, 7, and 3 respectively. TCAM #3 is selected due to its smallest counter value. Then, the IP address "166.103.142.1

$$\rho = \lambda \times T_s / K; \quad (4.1)$$

Let $\{Q_i\}_{i=1}^{\infty}$ be the (stochastic) process of the number of the IP addresses in the queue at time of the *i*-th arrival. And define:

$$\alpha_j(\rho) := \sum_{i+m=j-2} e^{\rho(i+1)}(-1)^m \rho^m (i+1)^m / m! \, (j \geq 2). \quad (4.2)$$

Then the parameters of the queue are given by [13-14]
The Loss Probability (Blocking Probability) and Throughout:

$$P_L := P(Q = n) = \frac{1 + (\rho - 1)\alpha_n(\rho)}{1 + \rho\alpha_n(\rho)}; \quad (4.3a)$$

$$Throughput^{vii} = \rho(1 - P_L); \quad (4.3b)$$

The Server Utilization (Probability of Non-empty Queue):

$$U := P(Q > 0) = \frac{\rho\alpha_n(\rho)}{1 + \rho\alpha_n(\rho)}; \quad (4.4)$$

The Expected Response Time (Queue Waiting Time):

$$R = \frac{n + (n\rho - 1)\alpha_n(\rho) - 1 - \sum_{j=2}^{n-1} \alpha_j(\rho)}{\rho\mu\alpha_n(\rho)}; \quad (4.5)$$

The comparison of the parameters of the four M/D/1/n queues with different "n" value (buffer depth) is shown in Fig. 7. Trading off between the loss probability and the response time, we choose n=10 as a typical value of buffer depth for each chip in our mechanism.

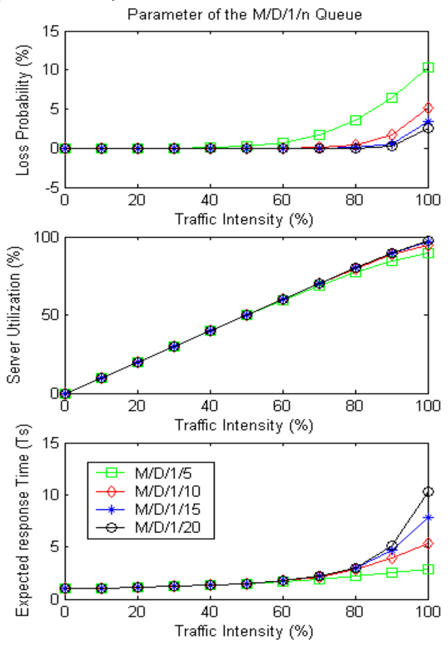

Figure 7. Comparison of the four M/D/1/n queues with different "n" value.

[vii] Throughput of a single server. The throughput of the system should be *K* times of this.

B. Simulation Results

In addition to the theoretical analysis of the algorithm we have run a series of experiments and simulations to measure its performance and adaptability on different types of traffic load distributions. In the case of four TCAMs with a buffer depth n=10 for each, we use two different traffic load distributions among the ID groups to simulate of our proposed scheme. The results are given in Fig. 8.

When the traffic is more evenly distributed (Case #1, the top two diagrams in Fig. 8), we learn from the simulation result that the introduction of storing redundancy only improves the throughput slightly. The load-balance-based memory organization has already restricted the loss (block) rate within 5%. On the other hand, in the case of non-even distribution (Case #2, the bottom two diagrams in Fig. 8), the storing redundancy improves the lookup throughput distinctly when the system is heavily loaded, even though the redundancy rate is as low as 1.25. In both cases, a redundancy rate of 1.25 guarantees the throughput to be nearly 100%, which means that when four TCAMs work in parallel, the proposed scheme promotes the lookup throughput by a factor of 4 and only 25% more memory space is required.

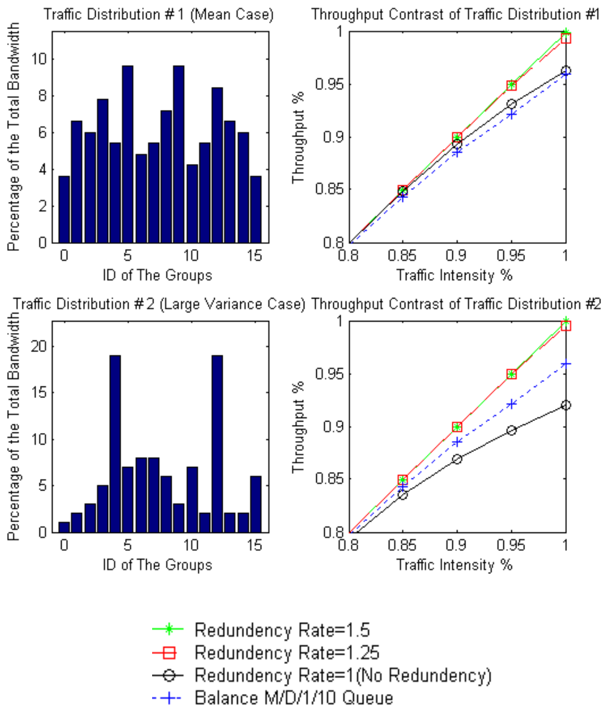

Figure 8. Comparison of the throughput when using different redundancyrate and applying different traffic distribution.

In order to measure the stability and adaptability of the proposed scheme when the traffic distribution varies apart from the original one over time, we run the following simulations with the redundancy rate of 1.25.

1) Assuming that the forwarding table is constructed from the traffic distribution given in Case #2 shown in Fig. 8, we apply the lookup traffic with the distribution given in Case #1;
2) Assuming that the forwarding table is constructed from the traffic distribution given in Case #1 shown in Fig. 8, we apply the lookup traffic with the distribution given in Case #2;
3) Assuming that the forwarding table is constructed from the traffic distribution given in Case #1, we apply strictly even-distributed lookup traffic.

Fig. 9 shows the results of the three simulations. Although the traffic distribution varies a lot, the lookup throughput drops less than 5%, meaning that the proposed scheme is not sensitive to the variation of traffic distribution. In fact, the adaptive load-balancing mechanism plays an important role in such cases.

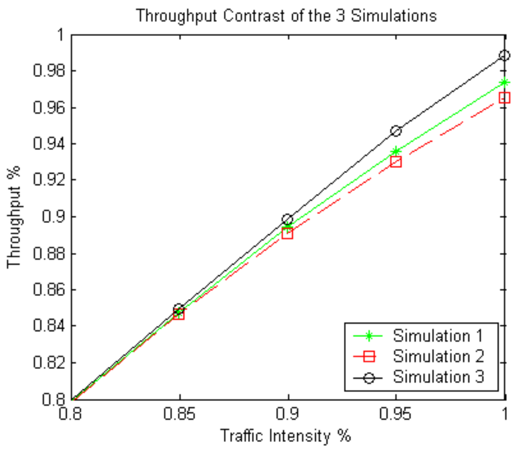

Figure 9. Results of the three simulations.

The input queue and the ordering logic also introduce some processing latency to the incoming IP addresses. Fig. 10 shows simulation results of the queuing and the entire processing (ordering logic included) latency of our mechanism.

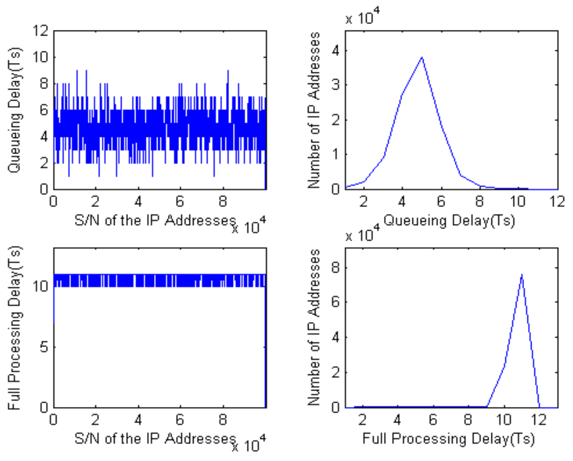

Figure 10. Processing latency and latency distribution of the incoming IP addresses.

The entire processing delay is between 9 to 12 T_s (service cycles). If a 133MHz TCAM is used, the delay is around 60ns to 90ns, which is acceptable. We also find that the jitter of the entire processing latency is small, which is critical for hardware implementation.

V. Conclusions

Increasing the lookup throughput and reducing the power consumption of the TCAM-based lookup engine are the two issues discussed in this paper. To distinctly increase the lookup speed and meet the demand of the next-generation terabit routers, parallel mechanism using multiple chips should be deployed. Two topics of applying the parallelism are how to allocate route prefixes evenly and how to balance the lookup traffic among the TCAMs. On the other hand, the power consumption of the TCAM-based lookup engine can be lowered down by reducing the number of entries triggered during a lookup operation. Multi-chip structure and chip partitioning technique are efficient method for this purpose.

In this paper, we proposed a scheme to address both problems. We give a simple but efficient TCAM table partitioning method and present a distributed memory organization based on the study of real-life route tables. We also develop a mathematical model of the problem on load-balance-based TCAM table construction and devise a greedy algorithm to solve it. Based on the performance study, given 25% more memory space, the proposed scheme increases the lookup throughput by a factor of 4 (compared with the single chip scheme) and significantly cut down the power consumption. Table IV shows the parameters of a real implementation of the scheme.

TABLE IV. A Real-life Example

Feature	Parameter
Number of TCAM Chips	4
Chip Size	256K*36b=9Mbit
Number of Partitions in Each Chip	8
Max. Number of Route Entries Supporting	819.2K
Redundancy Rate	1.25
Working Frequency of TCAM	133MHz
Max. Lookup Throughput	533Mpps
Max. Power Consumption*	$8\times 4/8$ =4 Watts
High Speed On-chip Cache Requirement	(10*4+40)×32bits=320Bytes
Avg. Processing Latency	75ns
Number of Data Buses	4

*: For each chip, the maximum power consumption (all the entries are enabled for search) is 8 Watts.

VI. Acknowledgement

The authors would like thank Mr. Guansong Zhang and Dr. H. Jonathan Chao for their valuable comments on this paper.

VII. Reference

[1] P. Gupta, S. Lin, and N. McKeown, “Routing lookups in hardware at memory access speed”, *IEEE INFOCOM’98,* San Francisco, April 1998, pp. 1240-1247.

[2] Nen-Fu Huang, Shi-Ming Zhao, and Jen-Yi Pan, “A fast IP routing lookup scheme for gigabits switching routers”, *IEEE INFOCOM’99*, vol.3, pp. 1429 –1436.

[3] Daxiao Yu, B.C. Smith, and B. Wei, “Forwarding engine for fast routing lookups and updates”, *IEEE GLOBECOM’99*, vol.2, pp. 1556 –1564.

[4] CYRESS. http://www.cypress.com/.

[5] IDT. http://www.idt.com/products/.

[6] Netlogic Microsystems. http://www.netlogicmicro.com/.

[7] H. Liu, “Routing Table Compaction in Ternary CAM”, *IEEE Micro*, 22(1):58-64, January-February 2002.

[8] F. Zane, G. Narlikar, A. Basu, “CoolCAMs: Power-Efficient TCAMs for Forwarding Engines”, *IEEE INFOCOM’03*, San Francisco, http://www.ieee-infocom.org/2003/papers/02_01.PDF.

[9] A. J. McAuley and P. Francis, “Fast Routing Table Lookup Using CAMs”, *IEEE INFOCOM’93*, San Francisco, CA, USA,March 1993, pp. 1383-1391.

[10] R. Panigrahy, S. Sharma, “Reducing TCAM Power Consumption and Increasing Throughput”, *Proceedings of HotI’02*, Stanford, California, USA, pp. 107.

[11] D. Shah and P. Gupta, “Fast Updating Algorithms for TCAMs”, *IEEE Micro*, 21(1):36-47, January-February 2001.

[12] K. Zheng,, H.B. Lu, and B. Liu, “A Parallel IP Lookup Algorithm for Terabit Router”, *Proceeding of ICCT*, April 9-11,2003, Beijing, China, pp.478-481.

[13] K.S. Trivedi, *Probability & Statistics with Reliability, Queuing, and Computer Science Applications*, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632.

[14] S. Alouf, P. Nain, D. Towsley, “Inferring Network Characteristics via Moment-Based Estimators”, http://www-sop.inria.fr/mistral/personnel/Sara.Alouf/Publications/infer.ps.gz.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms,* The MIT Press, pp. 984, pp. 1013-1017.

VIII. Appendix

In this section, we aim at proofing that the **Load-Balance-Based Table Construction Problem** (short for the **LBBTC** problem in the later part of this section) mentioned in Section II is an NP-hard problem. We first introduce the **Subset-Sum** problem and the **Average-Division** problem. The first one is a classic NP-Complete problem, and the second one can also be proven to be NP-Complete. We then demonstrate that the **LBBTC** problem is a NP-hard problem by showing the reduction from the **Average-Division** problem to the **LBBTC** problem.

Terminology:

We say that a language L_1 is ***polynomial-time reducible*** to a language L_2, written $L_1 \leq_P L_2$, if there exists a polynomial-time computable function $f:\{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$, $x \in L_1 \Leftrightarrow f(x) \in L_2$. [15]

The Subset-Sum Problem:

Given a finite set $S=\{1,2,...,n\}$, the weight function $w: S \to Z$, and the target $t \in Z$, we ask whether there is a subset $S' \subseteq S$ that can satisfy $\sum_{x\in S'} w(x) = t$.

We define the **Subset-Sum** problem as a language:

SUBSET_SUM: =

$\{<S,w,t>:$ $S \subset N$,

w is a function from $N \to Z$,

$t \in Z$, there exists a subset $S' \subseteq S$ such that $\sum_{x\in S'} w(x) = t$ $\}$

The Average-Division Problem:

Given a finite set $S=\{1,2..n\}$, and the weight function $w: S \to Z$, we ask whether there is a subset $S' \subseteq S$ can satisfy

$$\sum_{x\in S'} w(x) = \frac{1}{2}\sum_{x\in S} w(x) \cdot$$

We define the **Average-Division** problem as a language:

AVG_DIV: =

$\{<S,w>:$ $S \subset N$,

w is a function from $N \to Z$,

there exists a subset $S' \subseteq S$ such that $\sum_{x\in S'} w(x) = \frac{1}{2}\sum_{x\in S} w(x)$ $\}$

Lemma A:

The **Subset-Sum** problem is NP-complete. [15]

Lemma B:

The **Average-Division** problem is NP-complete.

Proof:

To show that **AVG_DIV** is in NP class, for an instance $<S,w>$ of the problem, we let the subset be the certificate. Checking whether $\sum_{x\in S'} w(x) = \frac{1}{2}\sum_{x\in S} w(x)$ can be accomplished by a verification algorithm in polynomial time.

We then show that **SUBSET_SUM** $\leq_P$ **AVG_DIV**, namely we show that **SUBSET_SUM** can be reduced to **AVG_DIV** in polynomial-time. Let $<S,w,t>$ be an instance of **SUBSET_SUM**, we construct an instance $<S_1,w_1>$ of **AVG_DIV** as follows:

Let $Sum = \sum_{x\in S} w(x)$;

Let $S_1 := S \cup \{n+1\} = \{1,2...n+1\}$;

$$w_1(x) = \begin{cases} w(x) ....x \in S \\ 2t - Sum....x = n+1 \end{cases}.$$

We now show that there exists a subset $S' \subseteq S$ that satisfies $\sum_{x\in S'} w(x) = t$ *if and only if* there exists a subset $S_1' \subseteq S_1$ that satisfies $\sum_{x\in S_1'} w_1(x) = \frac{1}{2}\sum_{x\in S_1} w_1(x)$.

It is clear that $\frac{1}{2}\sum_{x\in S_1} w_1(x) = t$ according to the definition above. If there exists a subset $S_1' \subseteq S_1$ and S_1' satisfies

$$\sum_{x\in S_1'} w_1(x) = \frac{1}{2}\sum_{x\in S_1} w_1(x) \Leftrightarrow \sum_{x\in S_1'} w_1(x) = \sum_{x\in S_1 - S_1'} w_1(x) = t;$$

Then we let $S' = \begin{cases} S_1' \ldots\ldots\ldots\ldots n+1 \notin S_1' \\ S_1 - S_1' \ldots\ldots n+1 \in S_1' \end{cases}$,

and because $n+1 \notin S'$, it is obvious that $S' \subseteq S$ and $\sum_{x \in S'} w(x) = \sum_{x \in S'} w_1(x) = t$; on the other hand, if there exists a subset $S' \subseteq S$ and satisfies $\sum_{x \in S'} w(x) = t$, then we let $S_1' = S'$, therefore,

$\sum_{x \in S_1'} w_1(x) = \sum_{x \in S_1'} w(x) = t = \frac{1}{2} \sum_{x \in S_1} w_1(x)$,
and $S_1' = S' \subseteq S = S_1$.

So **AVG_DIV** is NP-Complete.

Theorem 2:

The **LBBTC** Problem is NP-hard.

Proof:

We first introduce the decision problem of the **LBBTC** Problem and define it as a language, and then give the reduction from the **AVG_DIV** to the **LBBTC** Problem.

The Decision Problem of the LBBTC Problem:

Given the set of ID group $S = \{1,2,...,2^P\}$, the storing redundancy rate of the whole mechanism *Rd*, the number of the chips *K*, the tolerance of the discrepancy of the table sizes *T*, the traffic distribution among the ID groups $D\_id[j](j \in S)$, and a real number *q*. We ask whether there exists an allocation scheme $\{Q_i\}_{i=1}^K$ that can satisfy the following restrictions:

$$Q_j \subseteq S, j = 1,2,...,K, \ \bigcup_j Q_j = S; \qquad \text{(I)}$$

$$\left\| Q_i \right| - \left| Q_j \right\| \le T \quad (\forall 0 < i \le K, 0 < j \le K); \qquad \text{(II)}$$

$$BOOL(i,j) := \begin{cases} 1 \ldots\ldots\ldots j \in Q_i; \\ 0 \ldots\ldots\ldots j \notin Q_i; \end{cases}$$

$$G[j] := \sum_{i=1}^k BOOL(i,j) \quad (j \in S);$$

$$\sum_{j \in S} G[j] \le 2^P \times Rd \qquad (j \in S); \qquad \text{(III)}$$

$$1 \le G[j] \le K, \ G[j] \in Z^+ \quad (j \in S); \qquad \text{(IV)}$$

$$D[i] := \sum_{j \in Qi} D\_id[j] / G[j] \quad (i = 1,..., K); \qquad \text{(V)}$$

$$\underset{i}{Max}(D[i]) - \underset{i}{Min}(D[i]) \le q. \qquad \text{(VI)}$$

We define the **LBBTC** problem as a language:

LBBTC: =
$\{< S, K, Rd, T, D\_id, q >$:

$S = \{1,2,3,...,2^P\}$ is the set of the ID groups,

$K \in N$ is the number of the chips,

$Rd \in R$ is the redundancy rate of the whole mechanism,

$T \in N$ is the tolerance of the discrepancy of the table sizes,

D_id is a function (mapping) from $S \to R$, is the traffic distribution among the ID groups,

$q \in R$, and

there exists an allocation scheme $\{Q_i\}_{i=1}^K$ that satisfies the six restrictions given in the **LBBTC** decision problem.}

We now show that **AVG_DIV** $\le_P$ **LBBTC.** Namely we show that **AVG_DIV** can be reduced to **LBBTC**. Let $< S_1, w >$ be an instance of **AVG_DIV**, we construct an instance $< S, K, Rd, T, D\_id, q >$ of **LBBTC** as follows:

Let $S = S_1$, $D\_id[i] = w(i)(i \in S_1)$, $K = 2, Rd = 1, T \to \infty, q = 0$, then restriction (II) is now released, and using restriction (III) and (IV), we get that $G[j] = 1(j = 1,2), Q_1 \cap Q_2 = \phi$. And the restrictions are re-constructed as follow:

$$Q_1 \cup Q_2 = S_1; \qquad (1)$$

$$Q_1 \cap Q_2 = \phi; \qquad (2)$$

$$D[j] := \sum_{x \in Qj} w(x) \qquad (j = 1,2); \qquad (3)$$

$$\underset{j}{Max}(D[j]) - \underset{j}{Min}(D[j]) \le 0$$

$$\Leftrightarrow \quad D[1] = D[2] = \frac{1}{2} \sum_{x \in S_1} w(x). \qquad (4)$$

We now show that $< S_1, w > \in$ **AVG_DIV** *if and only if* $< S_1, 2, 1, \infty, w, 0 > \in$ **LBBTC** as follow**:**

If there exists an allocation scheme $\{Q_1, Q_2\}$ satisfying restriction (1-4), then we let $S_1' = Q_1$, and according to restriction (3) and (4), $\sum_{x \in S_1'} w(x) = \frac{1}{2} \sum_{x \in S_1} w(x)$. On the other hand, if there exists a subset $S_1' \subseteq S_1$ which satisfies $\sum_{x \in S_1'} w(x) = \frac{1}{2} \sum_{x \in S_1} w(x)$, then when we let $Q_1 = S_1'$ and $Q_2 = S_1 - S_1'$, it is clear that restriction (1-4) can all be satisfied.

So **AVG_DIV** can be reduced to **LBBTC**. Therefore, **LBBTC** is NP-hard.