Power Efficient IP Lookup with Supernode Caching

Lu Peng, Wencheng Lu* and Lide Duan

Department of Electrical & Computer Engineering
Louisiana State University
Baton Rouge, LA 70803
{lpeng, lduan1}@lsu.edu

*Dept. of Computer & Information Science & Engineering
University of Florida
Gainesville, FL 32611
wlu@cise.ufl.edu

***Abstract*— In this paper, we propose a novel supernode caching scheme to reduce IP lookup latencies and energy consumption in network processors. In stead of using an expensive TCAM based scheme, we implement a set associative SRAM based cache. We organize the IP routing table as a supernode tree (a tree bitmap structure) [5]. We add a small supernode cache in-between the processor and the low level memory containing the IP routing table in a tree structure. The supernode cache stores recently visited supernodes of the longest matched prefixes in the IP routing tree. A supernode hitting in the cache reduces the number of accesses to the low level memory, leading to a fast IP lookup. According to our simulations, up to 72% memory accesses can be avoided by a 128KB supernode cache for the selected three trace files. Average supernode cache miss ratio is as low as 4%. Compared to a TCAM with the same size, 77% of energy consumption can be reduced.**

Keywords-Network processor; Tree bitmap; IP lookup; Supernode; Caching;

I. INTRODUCTION

Packet routing is a critical function of network processors. An IP router determines the next network hop of incoming IP packets by destination addresses inside the packets. A widely used algorithm for IP lookup is Longest Prefix Matching (LPM). The adoption of a technique Classless Inter-Domain Routing (CIDR) [11] had made address allocation more efficient. In an IP router with CIDR, a <route prefix, prefix length> pair denotes an IP route, where the prefix length is between 1 and 32 bits. For every incoming packet, the router determines the next network hop in two steps: First, a set of routes with prefixes that match the beginning of the incoming packet's IP destination address are identified. Second, the IP route with the longest prefix among this set of routes is selected to route the incoming IP packet.

IP routing table organization and storage is a challenging design problem for routers with increasingly large tables. Many commercial network processors [6][8][10] achieve wire speed IP routing table lookup through high speed memories such as Ternary Content Addressable Memories (TCAMs) and specialized hardware. TCAMs have an additional "don't care" bit for every tag bit. When the "don't care" bit is set the tag bit becomes a wildcard and matches anything. TCAM's fully-associative organization makes it parallelly search all the routes simultaneously, leading to low access latency. However, its high cost and high power consumption [15][9] hamper TCAM being widely used.

Recently, researchers proposed the replacement of TCAMs by relative less expensive SRAMs. With well organizations, SRAMs can also achieve high throughput and low latency for IP routing table lookup [4][9][12]. In this paper, we propose a supernode based caching scheme to efficiently reduce IP lookup latency in network processors. A supernode is a tree bitmap node proposed in [5]. In a 32-level binary tree, we represent it by an 8-level supernode tree if we compress all 4-level subtrees, whose roots are at a level that is a multiple of 4 (level 0, 4, .. 28), to be supernodes. We add a small supernode cache in-between the processor and the low level memory containing the IP routing table in a tree structure. The supernode cache stores recently visited supernodes of the longest matched prefixes in the IP routing tree. A supernode hitting in the cache reduces the number of accesses to the low level memory, leading to a fast IP lookup.

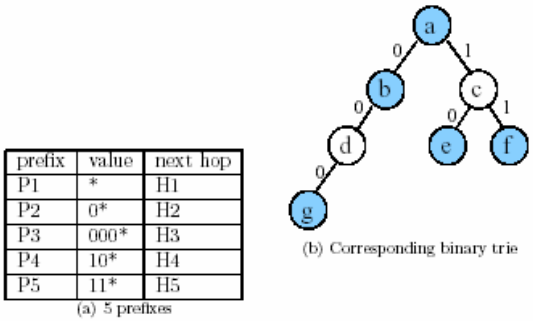
In our simulation, we compared the proposed supernode caching scheme with another two caches: a simple set-associative IP address cache and a fully-associative TCAM. Several results can be summarized from our experiments: (1) Average 69%, up to 72%, of total memory accesses can be avoided by using a small 128KB supernode cache for the selected three IP trace files. (2) A 128KB of our proposed supernode cache outperforms a same size of set-associative IP address cache 34% in the average number of memory accesses. (3) Compared to a TCAM with the same size, the proposed supernode cache saves 77% of energy consumption.

The left of this paper is organized as follows. Section 2 introduces related concept of the tree bitmap structure. Section 3 explains the proposed supernode caching scheme. Section 4 lists our experiment results. Section 5 makes a conclusion.

II. RELATED WORK

Many of the data structures developed for the representation of a forwarding table are based on the *binary trie* structure [7]. A binary trie is a binary tree structure in which each node has a data field and two children fields. Branching is done based on the bits in the search key. A left child branch is followed at a node at level *i* (the root is at level 0) if the *i*th bit of the search key (the leftmost bit of the search key is bit 0) is 0; otherwise a right child branch is followed. Level *i* nodes store prefixes whose length is *i* in their data fields. The node in which a prefix is to be stored is determined by doing a search using that prefix as key.

1930-529X/07/$25.00 © 2007 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2007 proceedings.



prefix	value	next hop
P1	*	H1
P2	0*	H2
P3	000*	H3
P4	10*	H4
P5	11*	H5

(a) 5 prefixes

(b) Corresponding binary trie

Figure 1: Prefixes and corresponding binary trie.

Figure 1(a) shows a set of 5 prefixes. The * shown at the right end of each prefix is used neither for the branching described above nor in the length computation. So, the length of *P2* is 1. Figure 1(b) shows the binary trie corresponding to this set of prefixes. Shaded nodes correspond to prefixes in the rule table and each contains the next hop for the associated prefix.

Tree bitmap (TBM) [5] has been proposed to improve the lookup performance of binary tries. In TBM we start with the binary trie for our forwarding table and partition this binary trie into subtries that have at most S levels each. Each partition is then represented as a (TBM) supernode. Figure 2 (a) shows a partitioning of the binary trie of Figure 2 (b) into 4 subtries W--Z that have 2 levels each. Although a full binary trie with S = 2 levels has 3 nodes, X has only 2 nodes and Y and Z have only one node each. Each partition is represented by a supernode (Figure 2 (b)) that has the following components:

1. A $(2^S - 1)$-bit bit map IBM (internal bitmap) that indicates whether each of the up to $2^S - 1$ nodes in the partition contains a prefix. The IBM is constructed by superimposing the partition nodes on a full binary trie that has *S* levels and traversing the nodes of this full binary trie in level order. For node W, the IBM is 110 indicating that the root and its left child have a prefix and the root's right child is either absent or has no prefix. The IBM for X is 010, which indicates that the left child of the root of X has a prefix and that the right child of the root is either absent or has no prefix (note that the root itself is always present and so a 0 in the leading position of an IBM indicates that the root has no prefix). The IBM's for Y and Z are both 100.

2. A 2^S-bit EBM (external bit map) that corresponds to the 2^S child pointers that the leaves of a full *S*-level binary trie has. As was the case for the IBM, we superimpose the nodes of the partition on a full binary trie that has *S* levels. Then we see which of the partition nodes has child pointers emanating from the leaves of the full binary trie. The EBM for W is 1011, which indicates that only the right child of the leftmost leaf of the full binary trie is null. The EBMs for X, Y and Z are 0000 indicating that the nodes of X, Y and Z have no children that are not included in X, Y, and Z, respectively. Each child pointer from a node in one partition to a node in another partition becomes a pointer from a supernode to another supercode. To reduce the space required for these inter-supernode pointers, the children supernodes of a supernode are

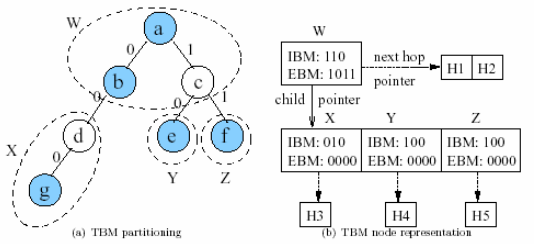


Figure 2: TBM for binary trie of Figure 1(b)

stored sequentially from left to right so that using the location of the first child and the size of a supernode, we can compute the location of any child supernode.

3. A child pointer that points to the location where the first child supernode is stored.

4. A pointer to a list *NH* of next-hop data for the prefixes in the partition. *NH* may have up to $2^S - 1$ entries. This list is created by traversing the partition nodes in level order. The *NH* list for W is *H1* and *H2*. The *NH* list for X is *H3*. While the *NH* pointer is part of the supernode, the *NH* list is not. The *NH* list is conveniently represented as an array.

The *NH* list (array) of a supernode is stored separate from the supernode itself and is accessed only when the longest matching prefix has been determined and we now wish to determine the next hop associated with this prefix. If we need *b* bits for a pointer, then a total of $2^{S+1} + 2b - 1$ bits (plus space for an *NH* list) are needed for each TBM supernode. Using the IBM, we can determine the longest matching prefix in a supernode; the EBM is used to determine whether we should move next to the first, second, etc. child of the current supernode. If a single memory access is sufficient to retrieve an entire supernode, we can move from one supernode to its child with a single access. The total number of memory accesses to search a supernode trie becomes the number of levels in the supernode trie plus 1 (to access the next hop for the longest matching prefix).

III. SUPERNODE CACHING

We compress the binary routing table tree into a supernode tree which is stored in low level memory. If a supernode corresponds to an 8-level subtree, a 32-level binary tree is compressed into a 4-level supernode tree. Assume that each supernode access takes one memory access, the maximum number of memory accesses for an IP lookup is five: it reads three supernodes plus the root nodes and searches the next hop for the longest matching prefix. When the root supernode is always held in cache, this number becomes 4. Obviously, maintaining a small cache will help to reduce the number of memory accesses.

Figure 3 illustrates an example of a supernode cache. In this example, the 32-level binary IP routing table is compressed into a 4-level supernode tree. Each supernode contains an 8-level subtree. The compressed 4-level supernode tree is stored in low level memory. We use a cache to store supernode addresses in low level memory. We store the root (level 1) and

1930-529X/07/$25.00 © 2007 IEEE

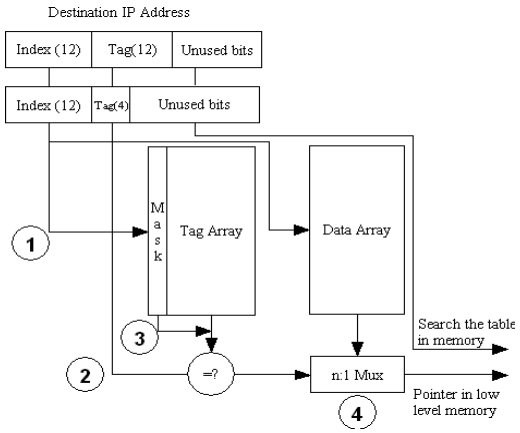


Figure 3. An example of a supernode cache

the second level supernode addresses containing 8 bits prefixes into a fully associative cache which is not shown in this figure. This is practical because there are at most $2^8 + 1 = 257$ addresses. For the third level and the fourth level supernodes, we reduce the search latency by introducing a set-associative cache. As shown by Figure 3, for a destination IP address, we search the corresponding set by its left most 12 bits (step 1). Bits 12 to 23 and bits 12 to 15 are used as tags for 24-bit and 16-bit IP prefixes (step 2). To implement the longest prefix matching, we identify the above two type of tags (12 bits and 4 bits) by a mask bit. For example, the mask bit 0 represents a 4-bit tag while a mask bit 1 means the tag has 12 bits. If both tags match, we select the longer one (step 3). After tag comparison, we select the corresponding entry in the cache's data array which is the address of the supernode in memory (step 4). A cache miss will result in cache update and replacement. We employ an LRU replacement policy. The search continues from the matched supernode in memory and proceeds downwards until the longest matching prefix is found. If a 4-level supernode is found in the cache, we only need one memory access for the next hop.

By leveraging the above cache design, we directly jump to a supernode in the search path of the bitmap tree, skipping over its ancestor supernodes along the path. However, we may fail to find the longest matching prefix if it exists in one of these ancestor supernodes. For example, consider the tree bitmap in Figure 2 and an incoming packet with the destination address 001*. We start the search of cache, if the address of supernode X is found, then the search continues at X and returns no match, though prefix 0* of binary node b should be returned. We solve this problem by pushing to the underlying binary root of each supernode a valid prefix from its lowest ancestor binary node. In this case, 0* of b is pushed down to d, and the search of X will successfully return 0* for the longest match of the destination address 0001*.

IV. Experiment Results

To evaluate the proposed supernode caching scheme, we download a routing table RS1221 from [2] and download three trace files from routers ipls, svl and upcb in the website [1]. In the following experiments, we collect statistics of first 1.5 million IP addresses whose longest prefix is larger than zero, i.e,

Trace File	IPLS	SVL	UPCB
Date	06/01/2004	03/18/2005	02/19/2004
Avg. Memory Access	4.28	4.73	4.64

Figure 4. Average memory access of 1.5 million IP addresses from each trace file

matching an inner node in the IP prefix tree. Totally, we implement four schemes with the longest prefix matching algorithm for IP routing: (1) without cache; (2) with an IP address cache; (3) with a TCAM; (4) with a supernode cache. For each scheme, we count the average memory access time. If there is a cache, we also measure miss ratios. In addition, we simulate energy consumption for each cache scheme.

In our experiments, we assume a 4-bit stride tree bitmap. Assuming that L denotes the number of steps taken to find the longest prefix node, the number of memory access for an IP lookup in the no cache scheme is $L/4 + 1$. In the second scheme, we design an IP address cache which contains the next hop information pointer in each entry. It can easily be implemented as a set associative cache by selecting part of the IP address bits as the set index and left bits as the tag. If an incoming IP address matches an entry in the IP address cache, it requires only one memory access to obtain the next hop. Otherwise, it needs $L/4 + 1$ memory access. In the third scheme, we assume that there exists a Tenary CAM. If an incoming IP address matches an entry in the TCAM, it requires only one memory access to obtain the next hop. Otherwise, it needs $L/4 + 1$ memory access. In the forth scheme, if the supernode cache hits, it takes $\lceil P - C \rceil / 4 + 1$ memory accesses. Here P denotes the length of the longest prefix and C denotes the length of supernode prefix hit in the cache. Otherwise, if the supernode cache misses, it requires $L/4 + 1$ memory accesses. In all of our experiments, we don't prefetch the routing tables into cache. Therefore, compulsory misses will be also included as misses.

Figure 4 shows the average numbers of memory accesses for the three selected trace files. Among the three files, IPLS has the smallest average number of memory accesses while SVL has the largest number. To understand the details, we further collect distributions of longest prefix matching (LPM) in Figure 5. Two observations can be made from it: (1) Most LPM hit the range from prefix length 8 to length 24. There is no matching for prefix length less than 8, therefore we do not show this range, while there are only a very few LPM hit prefixes longer than 24. (2) Three trace files show different distribution. IPLS has the largest group with prefix length 8 while SVL and UPCB have the largest two groups with length 16 and 17. One can expect that the proposed supernode caching scheme will have more performance benefits for SVL and UPCB than for IPLS because more relatively long supernode prefixes can be found from the supernode cache, resulting in less memory accesses.

1930-529X/07/$25.00 © 2007 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2007 proceedings.

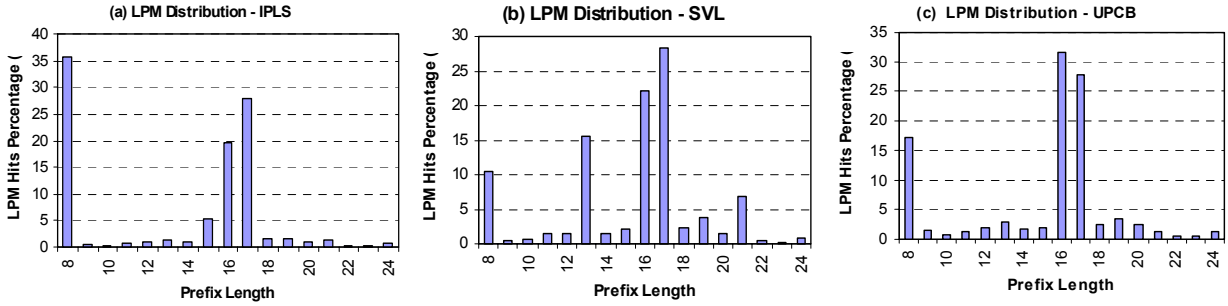


Figure 5. LPM distributions of the three trace files

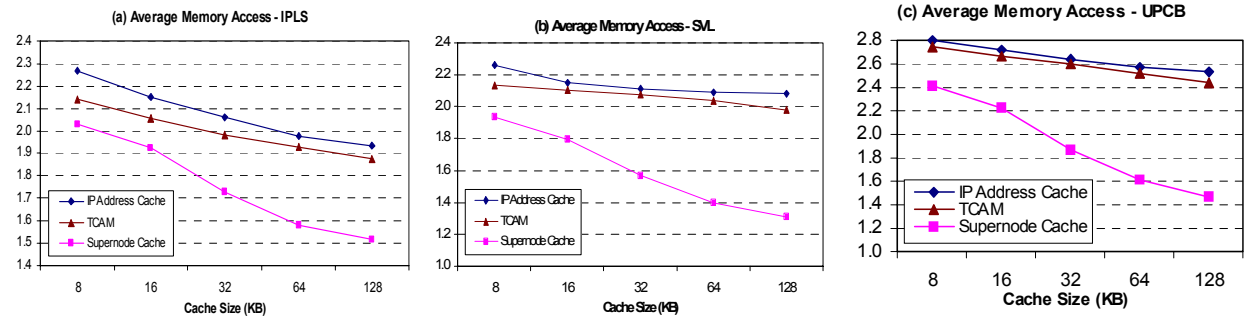


Figure 6. Average memory access of the three trace files.

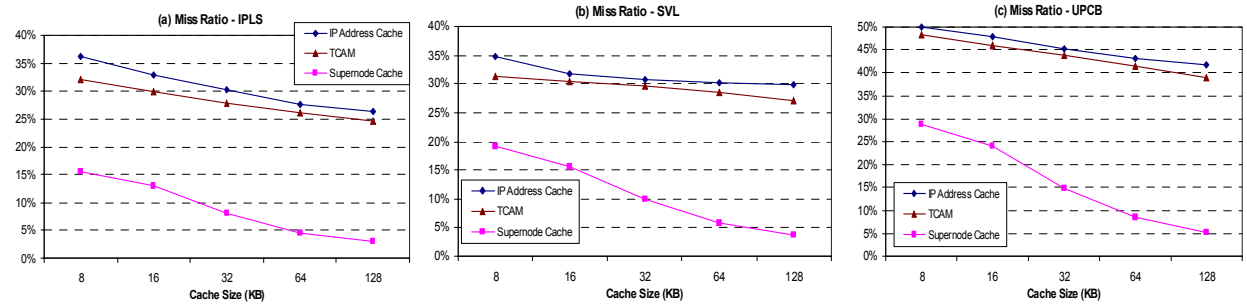


Figure 7. Miss ratios of three caching schemes for each trace file.

To illustrate the effect of supernode cache, we simulate different cache sizes ranging from 8KB to 128KB. We assume each cache entry has a four-byte width which can only store one unit because no spatial locality for IP addresses streams [4][13]. To implement the supernode cache, we set the Index field with 8 bits in Figure 3. There are five types of supernodes with different lengths will store in this cache: 12, 16, 20, 24 and 28. We use the mask field in the cache tag array to find the longest prefix matching. For all 8-bit supernodes, we assume that they are stored in a separate small cache. This is reasonable because the maximum size of this additional cache is 256 * 4 = 1KB. This implementation makes all supernode caches with fixed 256 sets. When the total cache size increases, we increase the set size instead of increasing the number of sets. To make comparison consistent, we also design the same total sizes and set sizes of IP address caches. In the combined cache scheme, we design half size as the IP address cache and another half as the supernode cache. For example, in a combined 32KB cache, the IP address cache is 16KB and the supernode cache is also 16KB.

Figure 6 illustrates average numbers of memory access with three caching schemes for each trace file. In general, all caching schemes reduce the average number of memory accesses. A 32KB IP address cache, TCAM and supernode cache reduce the average number of memory accesses from 4.73 to 2.11, 2.08 and 1.57, representing 55%, 56% and 67% reduction respectively, for SVL. In this case, the supernode cache outperforms the IP address cache 34% and the TCAM 32%. The average memory access reductions of the three caching schemes with 32KB are 50%, 51% and 62% separately for the selected three trace files. The supernode cache shows the best performance among the three caching schemes in all cases. When the cache size reaches 128KB, the supernode caching scheme's average memory access numbers for the three trace files are 1.51, 1.31 and 1.46, which means that 65%, 72% and 69% memory accesses are reduced. The average memory access reductions of the three caching schemes are 52%, 54% and 69% respectively for the selected three trace files with a 128KB cache size.

We also collect cache miss ratio information and present them in Figure 7. Several observations can be made: (1) the supernode cache has the smallest miss ratio, catching the

1930-529X/07/$25.00 © 2007 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2007 proceedings.

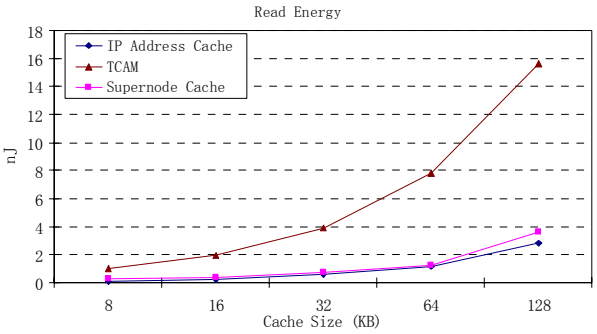


Figure 8. Read energy comparison of the three caching schemes.

strongest temporal locality in all cases. This is reasonable because a supernode, representing a subtree, will get reused if any node inside the subtree appears. The IP address cache and the TCAM will hit only if the same IP address recurs. For the three trace files, the miss ratios for a 32 KB supernode cache are 8%, 10% and 15% respectively. When the cache size reaches 128KB, the miss ratios for a supernode cache can be further reduced to 3%, 4% and 5% separately; (2) the slope of the IP address cache and the TCAM miss ratio lines are more flat than that of the supernode cache. This means that temporal locality of the IP address streams are limited. The possibility of recurrence of a supernode is larger than that of an IP address. Actually, this is the theory foundation of that a supernode cache outperforms an IP address cache.

To illustrate power efficiency of the proposed supernode cache, we simulate energy consumption of the three caching scheme. We use CACTI [14] to simulate the IP address cache and the supernode cache because they are set-associative cache. To make comparison fair, we also include power consumption of the small 1KB fully-associative cache which includes all 8-bits supernodes in the supernode cache scheme. According to Figure 5, the average activity of this small cache is about 20% for all three trace files. Therefore, in our simulation, we include 20% power consumption for this fully associative cache. We also simulate TCAM's power consumption using a recent model [3]. All three schemes are simulated under 0.18µm technology. Figure 8 depicts read energy which represents most of energy consumption of three caching schemes. From this figure, we can see that the supernode cache has a little higher energy consumption than the simple IP address cache because 20% additional searches fall into the small 1KB fully-associative 8-bit supernode cache. However, even in a 128KB cache size setting, the supernode cache consumes only 3.61nJ read energy. Compared with the TCAM's 15.64nJ read energy consumption, our proposed supernode caching scheme saves 77% energy consumption.

V. Conclusion

In this paper, we propose a novel supernode caching scheme to reduce IP lookup latencies and energy consumption in network processors. In stead of using an expensive TCAM based scheme, we implement a set associative SRAM based caching scheme. We organize the IP routing table as a tree bitmap structure. We add a small supernode cache in-between the processor and the low level memory containing the IP routing table in a tree structure. The supernode cache stores recently visited supernodes of the longest matched prefixes in the IP routing tree. A supernode hitting in the cache reduces the number of accesses to low level memory, leading to a fast IP lookup. According to our results, an average 69%, up to 72%, of total memory accesses can be avoided by using a small 128KB supernode cache for the selected three IP trace files. A 128KB of our proposed supernode cache outperforms a same size of set-associative IP address cache 34% in the average number of memory accesses. Compared to a TCAM with the same size, the proposed supernode cache saves 77% of energy consumption. The supernode cache works better for a trace file with larger groups LPM hits in relatively long prefixes. Our results also illustrate that the supernode cache catches stronger temporal locality than the other two cache schemes do.

Acknowledgment

This work is supported in part by the Louisiana Board of Regents grants NSF (2006)-Pfund-80 and LEQSF (2006-09)-RD-A-10, the Louisiana State University and an ORAU Ralph E. Powe Junior Faculty Enhancement Award. The authors thank Dr. Sartaj Sahni for his comments on the draft. Anonymous referees provide helpful comments.

References

[1] ftp://pma.nlanr.net/traces/

[2] http://bgp.potaroo.net/as1221/bgptable.txt

[3] B. Agrawal and T. Sherwood, "Modelling TCAM Power for Next Generation Network Devices", In the Proceedings of IEEE Intl. Symp. on Performance Analysis of Systems and Software (ISPASS-2006).

[4] T. Chiueh and P. Pradhan, "Cache Memory Design for Network Processors." In Proc.of the 6th International Symposium on High Performance Computer Architecture, pp. 409-418, Feb. 2000.

[5] W. Eatherton, G. Varghese, Z. Dittia, Tree bitmap: hardware/software IP lookups with incremental updates, Computer Communication Review, 34(2): 97-122, 2004.

[6] EZ Chip Network Processors. http://ezchip.com

[7] E. Horowitz, S.Sahni, and D.Mehta, Fundamentals of Data Structures in C++, W. H. Freeman, NY, 1995

[8] Intel IXP2850 Network Processor. http://www.intel.com/design/network/products/npfamily/ixp2850.htm

[9] S. Kaxrias and G. Keramidas, "IPStash: a Power-Efficient Memory Architecture for IP-lookup," In Proc. of the 36th International Symposium on Microarchitecture, Dec. 2003.

[10] Network and Communications ICs. www.agere.com/enterprise_metro_access/network_processors.html

[11] Y. Rekhter, T. Li, "An Architecture for IP Address Allocation with CIDR." RFC 1518, Sept. 1993.

[12] T. Sherwood, G. Varghese and B. Calder, "A Pipelined Memory Architecture for High Throughput Network Processors," In Proc. of the 30th Intl. Symp. on Computer Architecture (ISCA), Jun. 2003.

[13] B. Talbot, T. Sherwood, B. Lin, "IP Caching for Terabit Speed Routers." Globecom'99, pp. 1565-1569, Dec., 1999.

[14] D. Tarjan, S. Thoziyoor and N. P. Jouppi, CACTI 4.0 Technical Report, http://www.hpl.hp.com/techreports/2006/HPL-2006-86.pdf

[15] F. Zane, G. Narlikar, A. Basu, "CoolCAMs: Power-Efficient TCAMs for Forwarding Engines." IEEE INFOCOM, Apr. 2003.