Computer Communications 33 (2010) 390–397

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Expediating IP lookups with reduced power via TBM and SST supernode caching

Ying Zhang [a], Lu Peng [a,*], Wencheng Lu [b], Lide Duan [a], Suresh Rai [a]

[a] Department of Electrical & Computer Engineering, Louisiana State University, Baton Rouge, LA 70803, United States
[b] Department of Computer & Information Science & Engineering, University of Florida, Gainesville, FL 32611, United States

ARTICLE INFO

Article history:
Received 17 February 2009
Received in revised form 12 August 2009
Accepted 14 October 2009
Available online 27 October 2009

Keywords:
Tree bitmap (TBM)
Shape shifting trie (SST)
IP lookup
Supernode
Caching

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-ssociative SRAM based cache. We use two different algorithms, tree bitmap (TBM) and shape shifting trie (SST), to organize an IP routing table as a supernode tree composed of a group of 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. According to our simulations, up to 72% memory accesses can be avoided by a 128 KB TBM supernode cache for the selected three trace files, and up to 78% memory accesses can be reduced while using a same size of SST supernode cache. Average TBM and SST supernode cache miss ratios are as low as 4% and 7%, respectively. Compared to a TCAM with the same size, the TBM and SST supernode caches can both reduce 77% of energy consumption.

1. 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) [1] 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 [2,3,4] 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 [5,6] hamper TCAM being widely used.

IP caching has been extensively studied in [7,8,9], where caches are leveraged to provide a fast path for IP lookup to improve the average lookup time. 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 [6,10,11] . In this paper, we propose a supernode based caching scheme to efficiently reduce IP lookup latency in network processors. We utilize two different strategies, tree bitmap and shape shifting trie, to construct two types of supernode trees. Tree bitmap (TBM), which is proposed in [12], organizes an IP routing table to a regular shaped supernode tree. 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. On the other hand, shape shifting trie (SST) [13] generates a supernode tree composed of irregular shaped supernodes, which optimized the worst case IP address lookup performance. Lu and Sahni [14] further reduced memory requirement and lookup time for supernode tree. We add a small TBM or SST 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 [15], we demonstrated that a small cache for regular shaped TBMs can significantly reduce the memory accesses and power consumption. In

* Corresponding author.
E-mail addresses: yzhan29@lsu.edu (Y. Zhang), lpeng@lsu.edu (L. Peng), wlu@cise.ufl.edu (W. Lu), lduan1@lsu.edu (L. Duan), srai@lsu.edu (S. Rai).

0140-3664/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2009.10.006

this paper, we further extend the supernode cache to work for not only regular shaped TBMs, but irregular shaped SSTs. We also present more descriptions and experimental results in this version.

While worst case lookup performance is critical to deal with malicious users who attempts to flood network with packets, average lookup performance is another important metric for power consumption and general Internet router throughput. Hence, one category of literature [7,8,9] has been focusing on the average performance metrics. For network processor manufacturers, power dissipation is a first-order concern in processor design. Previous studies [16,17,18] show that persistent high power consumption tends to rapidly heat the processor, and thus largely degrade its reliability and lifetime due to the high temperature, which will introduce serious problems to the processor, such as wear-out of the critical hardware. Reducing average memory accesses is an efficient method to decrease power consumption and improve general Internet router throughput [19,20]. Therefore, we focus on reducing average power consumption and the average number of memory accesses because they can better contribute to a cool network processor.

In our simulation, we compared the TBM and the SST 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 128 KB tree bitmap supernode cache for the selected three IP trace files, and up to 78% memory accesses can be reduced while using a same size of shape shifting trie supernode cache. (2) A 128 KB of our proposed supernode cache outperforms a same size of set-associative IP address cache 34% in the average number of memory accesses while organizing the IP routing table as a tree bitmap. (3) Compared to a TCAM with the same size, both the TBM and SST 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 and the shape shifting trie. Section 3 explains the proposed supernode caching scheme. Section 4 lists our experiment results. Section 5 makes a conclusion. Through the paper, we sometimes use the term subtree to indicate a supernode.

2. Related work and background

Many of the data structures developed for the representation of a forwarding table are based on the *binary trie* structure [21]. 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.

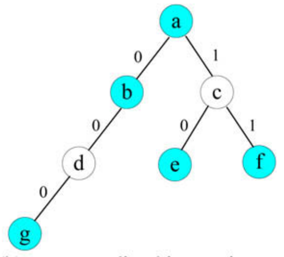
Fig. 1(a) shows a set of 5 prefixes. The * shown at the right end of each prefix is used neither for the branching described above nor in the length computation. So, the length of *P*2 is 1. Fig. 1(b) shows the binary trie corresponding to this set of prefixes. Shaded nodes correspond to prefixes in the rule table and each contains the next hop for the associated prefix. In this paper, we utilize two different optimized trees for the proposed supernode caching.

2.1. Tree bitmap (TBM)

Tree bitmap (TBM) [12] 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

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

Fig. 1. Prefixes and corresponding binary trie.

that have at most S levels each. Each partition is then represented as a (TBM) supernode. Fig. 2(a) shows a partitioning of the binary trie of Fig. 2(b) into 4 subtries W–Z that have 2 levels each. Although a full binary trie with S = 2 levels has three nodes, X has only 2 nodes and Y and Z have only one node each. Each partition is represented by a supernode (Fig. 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 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

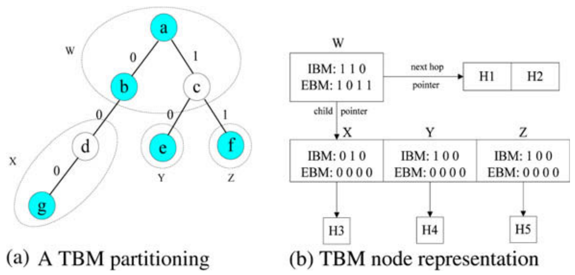


Fig. 2. TBM for binary trie of Fig. 1.