Routing Table Partitioning for Speedy Packet Lookups in Scalable Routers

Nian-Feng Tzeng, *Senior Member*, *IEEE*

Abstract—Most of the high-performance routers available commercially these days equip each of their line cards (LCs) with a forwarding engine (FE) to perform table lookups locally. This work introduces and evaluates a technique for speedy packet lookups, called SPAL, in such routers. The BGP routing table under SPAL is fragmented into subsets which constitute forwarding tables for different FEs so that the number of table entries in each FE drops as the router grows. This reduction in the forwarding table size drastically lowers the amount of SRAM (e.g., L3 data cache) required in each LC to hold the trie constructed according to the prefix matching algorithm. SPAL calls for caching the lookup result of a given IP address at its home LC (denoted by $\mathrm{LC_{ho}}$, using the LR-cache), such that the result can satisfy the lookup requests for the same address from not only $\mathrm{LC_{ho}}$, but also other LCs quickly. Our trace-driven simulation reveals that SPAL leads to improved mean lookup performance by a factor of at least 2.5 (or 4.3) for a router with three (or 16) LCs, if the LR-cache contains 4K blocks. SPAL achieves this significant improvement, while greatly lowering the SRAM (i.e., the L3 data cache plus the LR-cache combined) requirement in each LC and possibly shortening the worst-case lookup time (thanks to fewer memory accesses during longest-prefix matching search) when compared with a current router without partitioning the routing table. It promises good scalability (with respect to routing table growth) and exhibits a small mean lookup time per packet. With its ability to speed up packet lookup performance while lowering overall SRAM substantially, SPAL is ideally applicable to the new generation of scalable high-performance routers.

Index Terms—Caches, forwarding engines, interconnects, line cards, prefix matching search, routers, routing table lookups, tries.

1 Introduction

Rapid expansion of the Internet leads to sustained growth in the BGP routing tables held at backbone routers, and the table growth rate has expedited radically for the past three years [4], with certain routing tables now involving more than 140K prefixes (see AS1221, AS4637, and AS6447 in [4]). In fact, some backbone routers available commercially have provisions to accommodate 1 million or more prefixes, e.g., a Cisco 12000 Series Internet router may hold up to 1 million prefixes [10], while a Hitachi GR2000 Gigabit router supports up to 1.6 million prefixes [18]. As search in a routing/forwarding table is complex, usually based on *longest prefix matching search* to arrive at the most *specific* search result for a given IP address, it is common to organize prefixes as a tree-like structure called a *trie*, with its nodes either corresponding to prefixes or forming paths to prefixes [34], for effective search. The trie built under a chosen matching algorithm for a set of prefixes is highly desirable to fit within static RAM (SRAM) for good search performance. A rather large amount of SRAM is thus required for the forwarding engine (FE) at each line card (LC), in the form of an L3 data cache, increasing the LC cost markedly. Additionally, when IPv6 addressing is dealt with, the SRAM amount needed is likely to be several times higher, further in need of strategies for effectively containing the SRAM size.

Most commercial backbone routers carry out table lookups independently and concurrently at multiple FEs situated in different LCs, each of which houses one or multiple ports for external links to terminate. Examples of such routers include Cisco's 12000 Series routers [10], Juniper's T-Series backbone routers [22], and the Hitachi GR2000 Gigabit Router Series [18]. A full forwarding table with all prefixes is maintained in each LC of such a router, and a crossbar is adopted as the switching fabric for interconnecting its LCs (except for a small Hitachi GR2000 router with no more than four LCs, where a bus is used as the switching fabric). Every LC is equipped with one FE for conducting table lookups based on the longest-prefix matching algorithm implemented therein. To improve forwarding performance required by high-speed links operating up to the OC-768 (40 Gbps) rate in a router, one may employ a variety of approaches like enhanced routing/forwarding table lookup algorithms [11], [24], [35], [38], hardware-based lookup designs [17], [25], and hardware-assisted forwarding lookups [7], [16], [37]. This work deals with a technique for accelerating packet lookups in a scalable high-performance router with multiple LCs [6], as shown in Fig. 1.

The latency of a small crossbar switch has fallen considerably, resulting from a steady decline in the switching time of crossbars over the past decade due to the aggressive adoption of application specific Integrated Circuit to switch design and fabrication. Compared with then leading switches employed in the Mercury's RACE multicomputer system, known as the RACEway full crossbar with six ports and a switching time of 125 *ns* [29], later crossbars enjoy consistently lowered latencies, as evidenced by the Spider chip, which employs a fully multiplexed 6×6 crossbar and operates at a clock rate of 100 MHz [15], and by the Pericom's P15X1018 crossbar

• *The author is with the Center for Advanced Computer Studies, University of Louisiana at Lafayette, Lafayette, LA 70504.*
E-mail: tzeng@cacs.louisiana.edu.

Manuscript received 20 Oct. 2004; revised 18 May 2005; accepted 30 May 2005; published online 24 Mar. 2006.
Recommended for acceptance by J. Wu.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0256-1004.

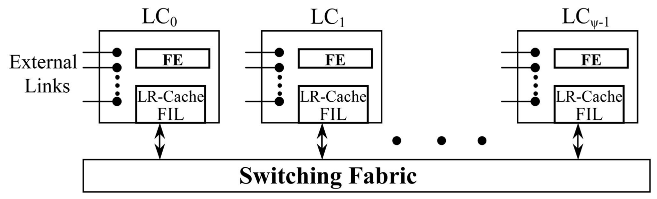


Fig. 1. A SPAL-based router, where FIL refers to fabric interface logics.

switch, which features 10 ports (of 18-bit width each) running at 133 MHz [27]. This latency reduction trend is likely to continue and one may expect to see a future crossbar with its switching time down to a few *ns*. With such crossbar switches, one may build a multistage-based switching fabric (according to a structure like, say, the Omega network) for interconnecting a moderate number of LCs in a router, with packet latency over the fabric being in the order of 10 *ns* or less.

The opportunity offered by fast switched crossbars (likely to facilitate a very low latency over a fabric built out of such crossbars) plus ever expanding BGP routing tables at backbone routers with a push to handle IPv6 addressing motivates this study on fragmenting a routing table into subsets of roughly equal sizes for LCs, so that the number of prefixes maintained in each forwarding table is a small fraction of the number of total prefixes kept in the routing table. Fragmentation is based on selected bit positions of prefixes in the routing table, and most prefixes are in single partitions after fragmentation. Each partition constitutes a forwarding table housed at one LC. The number of partitions (i.e., LCs) can be of any integer, not necessarily a power of 2. A small on-chip cache is introduced to each LC for holding the lookup results of destination addresses of the packets arriving at the LC. Caching lookup results permits subsequent lookup requests for same addresses to be satisfied immediately without resorting to (time-consuming) prefix matching by FEs (situated at either local LCs or remote ones). This caching takes merely 4K blocks at each LC to effectively reduce both traffic over the switching fabric and the load of requests for address lookups at each FE. Together, a hardware-assisted design for speedy packet lookups, called SPAL, in scalable high-performance routers is accomplished, with four salient features. First, SPAL drastically lowers the size of the trie (due to fewer prefixes) at each LC, making it possible to hold the whole trie (or a large portion of it) in the L3 data cache of the network processor at the LC for much faster matching algorithm execution. Second, the lookup result of an IP address cached at its home LC not only can satisfy forthcoming lookup requests from the home LC swiftly, but also can reply to the lookup requests from other LCs more quickly over a low-latency switching fabric than those LCs would otherwise have carried out prefix matching search themselves individually (taking hundreds of *ns*). Third, the cache introduced to an LC may also keep the lookup results obtained from other LCs (called remote LCs), so that requests for looking up same destination addresses of subsequent packets arriving at the LC can be satisfied locally more quickly, cutting down traffic over the switching fabric and reducing the request loads of those remote LCs. Finally, SPAL not only makes a router exhibit quicker mean lookups than its compatible router, but also enjoys good scalability (with FEs receiving relatively balanced load no matter how many LCs are involved), while possibly shortening the worst-case lookup time as well.

Lookup results obtained from remote LCs are held in the LR-cache of a local LC as well to 1) expedite subsequent lookup requests generated by the local LC for same addresses, 2) reduce traffic over the switching fabric, and 3) lessen request loads at remote LR-caches. The LR-cache equipped in each LC is on the same chip as the FIL (fabric interface logics, see Fig. 1), but it need not be very large to harvest almost full potential gains in performance: Our extensive simulation studies have indicated that a cache with some 4K blocks is usually adequate. Given 6 bytes in a block for IPv4 addressing, the amount of cache in each LC equals 24 Kbytes. (Note that IPv6 addressing requires 18 bytes per block.) On the other hand, the savings of SRAM resulting from a smaller trie in each LC after routing table partitioning typically amounts to hundreds of Kbytes, as will be detailed in Section 4, and the saving amount is expected to be far larger under IPv6. SPAL is an effective hardware-assisted design for fast packet lookups in a router, usually reducing its overall SRAM at each LC (including the off-chip L3 data cache plus the cache introduced to hold lookup results, called the LR-cache) tremendously.

When compared with its existing counterpart, a router under SPAL exhibits far quicker lookups. Our trace-driven simulation indicates that a SPAL-based router with three (or 16) LCs can forward, on an average, over 38 (or 347) million of packets per second, if each LR-cache involves 4K blocks, when the Lulea trie [11] is adopted for longest prefix matching. This average forwarding ability is 2.5 (or 4.3) times faster than that of an existing router, which keeps all prefixes of the routing table in each LC and has no LR-caches. Additionally, a SPAL-based router may shorten the worst-case lookup time, in comparison with a conventional router under the same longest prefix matching algorithm, as explained in Section 4.

While IP lookup traffic streams present different characteristics than the data streams of typical computing applications [26], recent work has demonstrated that a network processor with caches (of 4K blocks each and 4-way set associativity) has hit rates higher than 0.93 under traces collected via ESnet over the link (at the T3 rate) connecting the Brookhaven National Lab during 3-6 March 1998, leading to significant improvement in packet forwarding performance [7], [8]. Our extensive simulation studies using various recent traces collected over high-speed ports (including OC48c of Cisco GSR 12015 backbone routers during 14-27 August 2002 and also OC192c of the Internet backbone router at Indianapolis on 1 June 2004) available at the NLANR's PMA trace archive [28] confirmed the effectiveness of caches with respect to table lookups of IP traffic, requiring 4K blocks in each LR-cache to attain satisfactorily high performance under SPAL for all traces examined. These results contemn an earlier projection [38] that access locality in packet streams seemed to decrease and larger caches were required to achieve similar hit rates as time progressed. The projection is made according to simply a study in 1996 [26] that called for the cache of 5,000 entries to achieve the hit rate of 0.9, as opposed to nine entries for traffic observed in 1987. Fortunately, an independent investigation based on 1998 traces [7], [8] indicated the adequacy of 4K blocks in each cache to reach hit rates higher than 0.93. Our extensive simulation making use of publicly available 2002 and 2004 traces also concluded that the cache of 4K blocks is sufficient to enjoy high performance under SPAL. While the Internet has grown by more than 10 times from January 1996 (with 14 million hosts) to January 2003 (with 171.6 million hosts) [41], the locality of IP traffic over the Internet **does not drop** based on three separate, independent trace-driven simulation studies using 1998 traces, 2002 traces, and 2004 traces, perhaps due to the observed fact that a small percentage of flows between AS pairs (say, 9 percent) in the Internet accounts for a large percentage of total traffic (say, 90 percent) [13], [14]. This SPAL-based solution is believed to be equally effective in greatly quickening packet lookups for future Internet traffic as it is for past 1998 WorldCup traffic [39] as well as 2002 and 2004 backbone traffic [28] employed for our simulation.

For a given cache size, the larger a SPAL-based router is, the higher lookup performance it attains; this results mainly from fragmenting the set of prefixes (and, thus, the IP addresses) into more partitions based on SPAL, yielding better address space coverage (thanks to fewer prefixes) by each LR-cache. Due to its improved lookup performance and significant drop in the overall SRAM requirement particularly attractive for IPv6 addressing, our proposed SPAL technique is ideally applicable to future scalable high-performance routers.

2 Pertinent Work

This section first reviews earlier work related to packet lookups in routers, followed by a brief description of caching lookup results treated previously. A prior technique for table partitioning to enable parallel table search is then highlighted.

2.1 Packet Lookups

Packet lookups in routers can be expedited by various approaches, generally classified as software-based or hardware-based ones (depending upon if specific lookup hardware logics are required). A software-based approach often intends to either lower the memory requirement of a routing/forwarding table (so as to fit the table into fast SRAM, in the form of L3 data cache of the FE processor) or reduce the number of memory accesses during each lookup [11], [24], [35], [38]. As a variation of the (binary) trie obtained by compressing paths and with some modifications to support longest prefix matching, the *BSD trie* [34] was adopted in Berkeley Unix. Later, an enhanced trie implementation, called a *DP trie* (dynamic prefix trie), was considered to lower the average number of memory accesses upon search [12]. This DP trie yields a small code size and low storage requirements, confining the effects of random insertion and deletion operations to be local for rapid updates. Various algorithms resort to multiple-bit inspection at each search step (as opposed to single-bit inspection for the BSD trie and the DP trie), and the number of bits inspected at each time (called the *stride*) affects the search speed and the memory amount needed for keeping the trie [32]. If the stride is of 32 bits, for example, every search under IPv4 addressing takes just one memory access (being the fastest possible speed), but it requires a huge memory to store 2^{32} entries (being the largest possible size). Typically, a smaller stride leads to more memory accesses during search, but needs less memory.

It is simple to have a fixed stride for all the nodes at a given trie level. For more efficient memory utilization, however, variable strides can be adopted at the expense of more complicated implementation. For a given set of prefixes, the optimal strides which minimize the memory requirement and guarantee the worst-case number of memory accesses, can be derived using dynamic programming [35] for both fixed-stride and variable-stride cases. Separately, the Lulea algorithm constructs a 3-level compressed data structure, with the strides of 16, 8, and 8, respectively, for the first, second, and third levels [11]. The algorithm employs leaf pushing to avoid storing any prefix at an internal node of the resulting multiple-bit trie, so as to save memory. Another method replaces the largest full binary subtrie of a binary trie with a corresponding one-level multiple-bit subtrie recursively, starting with the root level, to produce an *LC-trie* (level-compressed trie) [24]. Search in an LC-trie requires an explicit comparison when arriving at a leaf to ensure a search match. A data structure for the routing table to quicken table updates has been considered recently [33]. While software-based approaches can be applied to 128-bit IPv6 prefixes, they often lead to far longer lookup times and bigger storage for the trie.

Hardware lookup designs have been proposed for high-speed routers under IPv4. In particular, a 2-level multibit trie with fixed strides was implemented in hardware to support IP lookups at the speed of memory accesses [17], with the first level realized by a table with 2^{24} entries addressed by the first 24 IP bits. Each table entry contains either forwarding information (for an IP prefix with length ≤ 24) or a pointer to the corresponding subtrie at the second level. Each subtrie in the second level contains 2^8 entries, and the total number of such subtries depends on the set of prefixes at hand. The access logic for this hardware design is simple, and the lookup time equals one memory access time (if pipelined properly). However, its memory requirement is huge (> 32 Mbytes). Distinct implementation schemes have been attempted later to lower the memory requirements, with proper hardware pipelines included to get one lookup per memory access [25].

Separately, network search engines using a combination of SRAM and RLDRAM II (reduced latency DRAM, with about one half of the cycle time of standard DDR SDRAMs) have been offered commercially by Xelerated Inc. [40] to act as coprocessors for IPv4 and IPv6 forwarding table lookups. Those search engines intend to free up computing resources provided by the network processors for running the router system and application codes, and they are an alternative to other search engines based on TCAMs (ternary content-addressable memories). While more expensive and consuming more power than their Xelerated counterparts, TCAM search engines (noticeably by NetLogic Microsystems [23] and Cypress Semiconductor Corp. [9]) usually deliver higher lookup rates. To make TCAM-based hardware design more attractive, different attempts have been made lately to lower its power consumption [30] and the storage requirements [31], or to achieve high throughput [42].

2.2 Caching Lookup Results

Caches are proven very effective in lowering memory access latencies and in avoiding repeated computation or transmission of identical information. A network processor equipped with hardware caches for capturing table lookup results has shown to improve overall packet forwarding performance significantly [7], [8], [16]. The caching algorithm described in [7] mapped IP addresses carefully to virtual addresses so as to make use of CPU caches (both L1 and L2) for fast lookups, reaching more than 80 million lookups per second according to detailed simulation on a 500 MHz Alpha processor with 16 Kbytes of L1 cache and 1 Mbytes of L2 cache (off-chip SRAM). Separately, a technique for improving the effective coverage of the IP address space has been considered by caching a *range* of contiguous IP addresses in each entry [8], [16]. It yields better performance when the address range cached in an entry is larger, since the address space covered by a given cache structure is then bigger. To this end, address range merging is adopted to get a large range. Two steps of address range merging were considered [8], [16]. The first step attempts to merge adjacent address ranges that share the same output interface in the prefix table into larger ranges, and then the ranges are aligned (to make them powers of two) before the minimum range size can be decided. This minimum range size dictates the effective address coverage improvement. The second step of range merging uses a properly chosen set of bits for indexing the prefixes so as to map originally nonadjacent prefixes to contiguous logical addresses for merging. A greedy bit selection algorithm was considered to minimize the total number of address ranges and the size differences among address ranges after mapping [8], [16]. Simulation results have confirmed that address range merging after proper mapping may improve caching efficiency (in terms of mean lookup time) markedly.

While some routing tables might give rise to the minimum range sizes larger than $2^0 = 1$, a backbone router tends to contain a growing number of prefix exceptions in its routing/forwarding table [19], [32], making the minimum range size equal to 1 and, thus, nullifying the potential benefit of the first step of range merging. According to prefix length distribution results [2], [32], a number of prefixes in the routing table of a typical backbone router is of length 32, rendering the minimum range granularity equal to 1. (Our two sets of prefixes used for this work both contain many prefixes of length 32 as well.) Thus, cache hit rates using the network processor cache after address range merging may not be as high as those presented earlier [8], [16].

2.3 Partitioning for Parallel Search

A different technique has been considered recently [1] for partitioning the routing table into subsets, which can then be searched in parallel. In the partitioning technique considered, all prefixes of a given length in the routing table are grouped into one partition. Clearly, the size of each partitioned subset varies considerably, e.g., the length of size 24 typically accounts for about 50 percent of all prefixes and many of the subsets (say, 10 or more subsets) each contain less than 1 percent of total prefixes in the routing table [2]. Unlike SPAL, the design in [1] keeps *all* partitioned subsets at each FE for search in parallel; table lookups of all packets arriving at an LC are performed locally in the LC, and no lookup result obtained by one LC may be shared by any other LC. In addition, the sizes of forwarding tables in LCs are not reduced when the number of LCs grows. No cache was introduced to LCs in the earlier design [1].

3 SPAL-Based Routers

Each LC in any currently existing router maintains the entire set of prefixes, whose size grows steadily over time, no matter how many LCs are within the router. As the trie size typically is proportional to the number of prefixes, this certainly calls for large SRAM (in the form of an L3 data cache of the FE processor, for example) in order to hold the trie for fast lookups, in particular, when IPv6 is concerned. The proposed SPAL aims to reduce the SRAM requirement while accelerating table lookups through fragmenting the BGP routing table into Ψ subsets (of roughly equal sizes), one for each LC (as its forwarding table) in a router with Ψ LCs. As depicted in Fig. 1, a low-latency switching fabric is employed to interconnect all LCs through fabric interface logics (FIL's), and the fabric can be a shared-bus (for a small Ψ), a crossbar, or a multistage-based structure (like the Omega network), among others. In this study, no emphasis on the fabric details will be placed, but the fabric latency (in terms of system cycles) is assumed to depend on the fabric size. Each LC also includes one small fast SRAM housed inside the FIL chip, referred to as the LR-cache, for holding the lookup results obtained by the local FE and by other remote FEs. Further discussion about the LR-cache will be given in Section 3.2.

3.1 Table Partitioning

Partitioning is done using appropriately chosen bits in IP prefixes, and a subset of routing table prefixes is referred to as an *ROT-partition*. For a router with Ψ LCs, the key decision on partitioning its routing table is to choose appropriate bits for yielding Ψ ROT-partitions in the most desirable way, namely, 1) each ROT-partition involving *as few prefixes as possible* and 2) the size difference between the largest ROT-partition and the smallest one being *minimum*. Any partitioning which satisfies these two criteria is deemed **optimum**. Let $\eta = \lceil \log_2 \Psi \rceil$, then the number of bits chosen for partitioning is η, where $\lceil x \rceil$ is the smallest integer $\geq x$. Note that Ψ *does not have to be a power of* 2 and can be any integer, say, $3, 5, 6, 7$, etc. The partitioning results of $\Psi = 3$ will be shown in Section 5. For easy explanation in the following, we consider simplified prefixes of up to 8 bits only, with the leftmost bit in a prefix denoted by b_0, the next bit by b_1, etc., despite that IP prefixes are actually sequences of up to 32 bits. The case of Ψ being a power of 2 is explained first, followed by a generalization to any arbitrary Ψ.

Given seven simplified prefixes in a routing table: $P_1 = 101*$, $P_2 = 1011*$, $P_3 = 01*$, $P_4 = 001110*$, $P_5 = 10010011$, $P_6 = 10011*$, $P_7 = 011001*$, if b_2 and b_4 are used for partitioning, we arrive at four ROT-partitions: $\{P_3, P_5\}$,

$\{P_3, P_6\}$, $\{P_1, P_2, P_3, P_7\}$, $\{P_1, P_2, P_3, P_4\}$, where the first partition corresponds to $b_2b_4 = 00$, the second one corresponds to $b_2b_4 = 01$, and the third (or fourth) one, to $b_2b_4 = 10$ (or 11). With this partitioning, κth ROT-partition resides in LC_κ, where $\kappa = 0, 1, 2, 3$. Any packet arriving at LC_κ is called a *local* packet, if b_2b_4 of its destination address equals κ since its lookup will be done by the local (accompanying) FE; otherwise, the packet is a remote one, with its *home* being $LC_h (h = b_2b_4)$. Each packet has one and only one *home LC*, which can be determined immediately upon arrival by examining the appropriate bit positions of the packet destination address. When missing in the LR-caches of their arrival LCs, nonlocal packets are delivered from their arrival LCs through the switching fabric to their respective home LCs for longest-prefix matching. Each such a packet is routed across the switching fabric using b2 and b4 of its destination address. Note that P_3 belongs to every partition, because both b_2 and b_4 of P_3 are "*" (which would match any IP address whose b_2 is 0 or 1 and whose b_4 is 0 or 1, requiring that P_3 should exist in each partition since a matched IP may arrive at any FE). Similarly, P_1 belongs to the third and the fourth partitions, as b_4 of P_1 is "*." On the other hand, if b_0 and b_4 are used for partitioning, we obtain the partitions of $\{P_3, P_7\}$, $\{P_3, P_4\}$, $\{P_1, P_2, P_5\}$, $\{P_1, P_2, P_6\}$, where each partition involves two or three prefixes. It is obvious that the latter partitioning is superior to the former one, based on the two criteria listed above.

For a given set of prefixes, a chosen bit (say b_v) separates prefixes into two subsets with $(\Phi_0 + \Phi_*)$ and $(\Phi_1 + \Phi_*)$ prefixes, respectively, where Φ_0 (or Φ_1) is the number of prefixes whose b_v bits equal "0" (or "1") and Φ_* is the number of prefixes with b_v bits being "*" (since these prefixes have to appear in both subsets). According to Criterion 1 above, we need to find out the bit b_v such that $(\Phi_0 + \Phi_1 + \Phi_* + \Phi_*) = \Phi + \Phi_*$ is smallest among all $0 \leq v \leq 31$, where Φ is the set size. This criterion is thus equivalent to locating the bit b_v which leads to a minimum Φ_*, ruling out a large v (say > 24), since the vast majority of prefixes in a routing table (e.g., more than 83 percent for the set of prefixes obtained from [2]) have length no more than 24, and b_v in a prefix is "*" for any v larger than the prefix length. Criterion 2 requires the search of bit b_v such that $|(\Phi_0 + \Phi_*) - (\Phi_1 + \Phi_*)| = |\Phi_0 - \Phi_1|$ is minimum for all $0 \leq v \leq 31$. When examining bit b_v, this criterion calls for ignoring prefixes with bit b_v = "*," counting only those with bit b_v being "0" or "1." Notice that for a single position bit, Criterion 1 considers only Φ_* of the position bit whereas Criterion 2 deals with Φ_0 and Φ_1 of the bit. Given Ψ and a prefix table, our partitioning algorithm *searches exhaustively* for η $(= \lceil \log_2 \Psi \rceil)$ control bits (out of 32 bits for IPv4) to get the optimum result following the aforementioned two criteria.

3.1.1 Generalization and Other Issues

Given an arbitrary integer Ψ, let Ψ_L equal $\lceil \Psi/2 \rceil$ and Ψ_R be $\Psi - \Psi_L$. The above partitioning approach is still applicable under such an integer value, with a modification to Criterion (2) needed: bit b_v chosen for partitioning satisfies that Φ_0/Φ_1 (or Φ_1/Φ_0) is as close to Ψ_L/Ψ_R as possible, if $|\Phi_0| \geq |\Phi_1|$ (or $|\Phi_0| < |\Phi_1|$), for all $0 \leq v \leq 31$. Each of those Ψ nodes is labeled by $\eta = \lceil \log_2 \Psi \rceil$ bits, determined according to the modified criterion shown by a binary tree given in Fig. 2. For $\Psi = 6$, Ψ_L is 3 and Ψ_R is also 3, as denoted by the two nodes in tree Level 1. Both Ψ_L and Ψ_R are further partitioned into two fragments each, as depicted by Level 2. A fragment with only 1 node becomes a tree leaf, whose label is determined by the path from the root to the leaf. For example, the rightmost leaf is labeled by 11*, meaning that every prefix with its 3 ($= \eta$ partitioning bits being 110 or 111 is homed at this LC. According to Fig. 2, those 6 ($= \Psi$) LCs are labeled, respectively, by 000, 001, 01*, 100, 101, and 11*. Note that if Ψ is a power of 2, Ψ_L/Ψ_R always equals 1.0 in each tree level, signifying Criterion 2 before generalization.

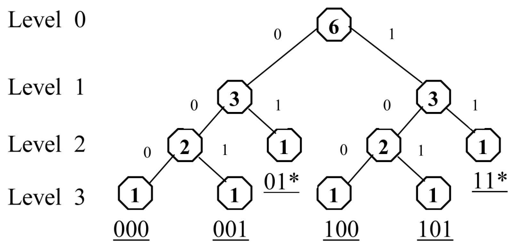


Fig. 2. The labels of LCs for $\Psi = 6$.

It should be noted that the partitioning mechanism devised for SPAL intends to minimize and equilibrate the number of prefixes held in each LC, according to the routing table of a router: The partitioning is irrespective of traffic over the router and is not meant to balance the load on LCs nor to optimize mean lookup performance. For a given routing table and a Ψ, a desirable partition is obtained according to the aforementioned method, and such a partition often gives rise to reasonably balanced load at LCs for all traces examined, regardless of the router size Ψ (as will be demonstrated in Fig. 9). Since the load of LCs is traffic dependent and changes dynamically, to realize a routing table partition that yields truly balanced load needs traffic profiling, which is not attempted in this work since it is expensive and difficult. Note also that while the routing table contents change over times as table updates take place, it is not necessary to carry out partitioning repeatedly. In practice, one may follow the partitioning method only once when a router gets its routing table established at the onset of its operation; the same partitioning should work almost equally well until any Ψ-partition grows excessively (and cannot be held in the SRAM of the associated LC effectively) due to table updates. This was confirmed by examining several routing tables to select the desired control bit positions according to our partitioning method before and after their entries being added or removed (reflecting table updates). For example, if the AS1221 routing table with 175,853 prefixes taken on 4 September 2004 [2] is partitioning into four fragments, the preferred partitioning bits determined by our partitioning method are at positions 14 and 25, giving rise to the fragments of sizes: 51,095, 50,886, 51,029, and 49,977 (see Section 4 for details). When the routing table at AS1221 was examined on 5 September 2004, the number of table entries became 176,043 (due to additions and withdrawals of entries, plus modifications to some entries in their next hop fields) [2], but the most desirable partitioning bit positions obtained by our partitioning method remain to be 14 and 25, yielding the fragments of sizes: 51,185, 50,996, 51,172, and 50,078. This partitioning result exhibits the total number of prefixes in the four fragments equal to 203,431 and the largest fragment size difference being 1,107. Additionally, the routing table AS1221 was found to have 176,424 entries on 6 September 2004 [2], and its preferred partitioning bits are still of positions 14 and 25, with four fragments of sizes: 51,336, 51,122, 51,306, and 50,285. When the AS1221 routing table is partitioned into 16 fragments, the same set of four partitioning bit positions is derived using our partitioning method for its table entries from 4 September to 6 September 2004. Consequently, the use of same partitioning bit positions even after a large number of table updates (over multiple days without recalculating a new set of bit positions) is generally seen to yield the best partitioning result.

An extra advantage resulting from routing table partitioning is that the address space coverage of each ROT-partition increases by approximately a factor of the number of partitions, since each ROT-partition is covered by one LR-cache. Given an IP lookup stream toward one LC, fewer prefixes in its forwarding table lead to better lookup performance (thanks to a higher hit rate). Partitioning results of two routing tables from different AS's will be demonstrated in Section 4.

3.1.2 Complexity Analysis

This complexity analysis considers partitioning the routing table into Ψ fragments, requiring to select $\eta\, (= \lceil \log_2 \Psi \rceil)$ control bits. Partitioning complexity involves three components:

1. selecting η control bit positions (out of 32 under IPv4) for partitioning, and under the selected η control bit positions:
2. deciding to which partition(s) each prefix in the routing table belongs, and
3. identifying the smallest and the largest partitions.

Clearly, there are totally $C(32,\eta) = 32!/(\eta! \times (32-\eta)!)$ choices for those η bit positions, reflecting the complexity component 1 above. For each of such choices, a prefix in the routing table belongs to one of the 3^η cases, determined by the values of those η bit positions being 0, 1, or *. Let those η bit positions be represented by $\mathrm{s}_0\mathrm{s}_1 \ldots \mathrm{s}_\mathrm{i} \ldots \mathrm{s}_{\eta-1}$, and the Ψ partitions be denoted by Ω_α, where $0 \le \alpha \le \Psi - 1$. A prefix belongs to Ω_α (with α expressed by the binary string of $\mathrm{b}_0\mathrm{b}_1 \ldots \mathrm{b}_\mathrm{i} \ldots \mathrm{b}_{\eta-1}$), if and only if $\mathrm{s}_\mathrm{i} = \mathrm{b}_\mathrm{i}$ or $\mathrm{s}_\mathrm{i} = *$; totally, its $\mathrm{s}_0\mathrm{s}_1 \ldots \mathrm{s}_\mathrm{i} \ldots \mathrm{s}_{\eta-1}$ can be in any of 2^η possible forms: $\mathrm{b}_0\mathrm{b}_1 \ldots \mathrm{b}_\mathrm{i} \ldots \mathrm{b}_{\eta-1}, *\mathrm{b}_1 \ldots \mathrm{b}_\mathrm{i} \ldots \mathrm{b}_{\eta-1}, \mathrm{b}_0 * \ldots \mathrm{b}_\mathrm{i} \ldots \mathrm{b}_{\eta-1}$, etc. For $\eta = 3$, as an example, Ω_2 contains all prefixes with $\mathrm{s}_0\mathrm{s}_1\mathrm{s}_2$ being 010, $*10$, $0{*}0$, $01*$, $**0$, $0**$, $*1*$, or $***$. On the other hand, a prefix with $\mathrm{s}_0\mathrm{s}_1\mathrm{s}_2 = *10$ belongs to Ω_6 as well. It is obvious that a prefix whose $\mathrm{s}_0\mathrm{s}_1 \ldots \mathrm{s}_\mathrm{i} \ldots \mathrm{s}_{\eta-1}$ has d $*'$s belongs to 2^d partitions, which can be determined immediately by $\mathrm{s}_0\mathrm{s}_1 \ldots \mathrm{s}_\mathrm{i} \ldots \mathrm{s}_{\eta-1}$ through setting each $*$-bit to be 0 or 1 individually for all d $*'$s. As a result, the complexity component (2) above is $O(2^d)$, which incurs when each prefix in the routing table is examined. Given N prefixes in the routing table, it leads to a time complexity of $\mathrm{O}(2^d) \times \mathrm{N}$ to go through all prefixes in the table.

The number of prefixes in each partition can be recorded in one variable, requiring Ψ variables totally. From those Ψ variables, one can get the total number of prefixes in all the partitions, needed for Criterion 1, and also the largest partition and the smallest one, needed for Criterion 2, by passing through those variables once. The time complexity component 3 thus is $\mathrm{O}(\Psi)$, bringing the time complexity for one set of η bit positions to be $\mathrm{O}((2^d) \times \mathrm{N}) + \mathrm{O}(\Psi) = \mathrm{O}(\Psi) \times \mathrm{N}$, as $\mathrm{O}(2^d) \le O(\Psi)$, for a routing table with N prefixes. Consequently, the overall time complexity of our partitioning algorithm is given by $\mathrm{C}(32,\eta) \times \mathrm{O}(\Psi) \times \mathrm{N}$. Since the same Ψ variables can be used repeatedly (after their values are reset to 0) for all $\mathrm{C}(32,\eta)$ choices of the η bit positions, the space complexity equals $\mathrm{O}(\Psi)$.

The derived time complexity expression indicates that our optimal partitioning strategy becomes more expensive as Ψ grows. In reality, the factor $\mathrm{C}(32,\eta)$ of the complexity expression can be reduced substantially because all selected control bit positions should be no larger than 24 according to Criterion 1, since the vast majority of prefixes in a routing table have length no more than 24, and should be no smaller than 8 according to Criterion 2, as those bit positions of prefixes in a routing table do not have similar numbers of 0 and 1. As a result, the time complexity in practice is reduced to $\mathrm{C}(16,\eta) \times \mathrm{O}(\Psi) \times \mathrm{N}$. If Ψ equals 16 (and, thus, $\eta = 4$) under N = 250K (which is larger than any existing routing table size), for example, the expression gives rise to 7.28×10^9. One may reduce the time complexity further by applying the two optimization criteria recursively, first to find one control bit b_v among $8 \le v \le 23$, and then to find a bit in each of the two subsets separately before deciding the bit for both subsets as the second control bit. Similarly, the third control bit is decided using the two criteria on the four subsets obtained using the two chosen control bits. This method leads to suboptimal partitioning with drastically reduced time complexity.

3.2 LR-Cache Operation and Organization

3.2.1 Principle of Operation

Typically, the routing table of a backbone router gets updated some 20 times per second on an average (and possibly as many as 100 times), leading to one table update in 50 - 10 *ms* [3], [32]. Once the routing table is updated, those changes should be reflected in the forwarding tables at all LCs. To ensure appropriate lookups, this study assumes that all entries in every LR-cache are *flushed after each table update*. (Note that while this simple flushing will not work as effectively as what will be demonstrated in Section 5.2, if the routing table is updated incrementally and very frequently, our simulation results provided later will unveil that the warm-up period for LR-caches to reach a specified hit rate after flushing is usually a small fraction of the total service time between two consecutive flushes.) After flushing, the availability status bit of each cache entry is set to the *invalid state*. Once an entry is chosen to hold a lookup result, its availability status bit is set to the *shared state*. Two types of lookup results may exist in LR-cache entries: results homed locally (LOC, obtained by the local FE) and results homed remotely (REM, obtained through remote FEs). An entry uses one bit, called the M (short for "mix") status bit, to indicate its LOC/REM status. This bit is necessary for implementing an efficient cache replacement mechanism, realizing a suitable "mix" of LOC entries and REM entries within each set (of cache entries) under a given cache organization (i.e., a given cache size, block size, and degree of set associativity) to achieve good performance.

When a lookup result is obtained through prefix matching by the FE in the home LC, the result is first placed in the LR-cache of the LC. If the cache has no free entry in the set of interest (decided by the destination IP address), one entry in the set has to be chosen for replacement. It chooses an entry with its M bit being REM (or LOC), if the total number of entries with M = REM (or LOC) in the set exceeds the predefined value, say 50 percent. The M status bit is examined first to decide the candidate block(s) to replace. (Note that hardware logics can be employed to make this decision instantly, since the number of blocks in each set does not have to be larger, say ≤ 4, to get nearly best performance, as revealed by our simulation results. Given four blocks in a set, for example, the logic can load the M bits of the four blocks to check them against 16 possible values: $0000_2, 0001_2, \ldots, 1111_2$, in parallel via the "XOR" operation.) A conventional replacement strategy (such as LRU, FIFO, or random) is then applied to the candidate block(s) to choose the one for eviction.

To lower the load of an FE and cut down traffic over the switching fabric, the LR-cache records a packet immediately when a miss occurs at the arrival LC. This early cache block recording prevents subsequent packets with the same destination from proceeding beyond the LR-cache, enhancing SPAL performance. Since this recorded cache entry is not complete until its corresponding reply is back, a status bit (waiting bit, i.e., W-bit) is added to the cache entry, with the bit set until its reply is back and fills the entry. When a forthcoming packet hits an LR-cache entry whose W-bit is set, the packet is stopped from proceeding forward and held in the waiting list associated with the incomplete entry. The packet is allowed to advance after the W-bit of the hit cache entry is cleared (by a reply). Once a reply comes back and hits the cache entry recorded earlier, the entry is completed with the lookup result (i.e., filling its Next_hop_LC# field) and its W-bit is cleared. The reply is then moved back to the arrival LC of the packet, if it is not local.

3.2.2 Cache Organization

An LR-cache is of on-chip SRAM and organized as a set-associative cache, with a block able to hold *one* lookup result (i.e., $<$ IP address, Next_hop_LC# $>$). The reason for a small block size is due to weak spatial locality of IP addresses in practice since the devices with contiguous IP addresses usually have little direct temporal correlation of network activities; a larger block size leads to poorer lookup performance because of decreased cache space utilization, as found by earlier studies [7], [16]. The degree of set associativity for LR-caches is chosen 4, and this choice leads to nearly best performance, according to an earlier work [37] as well as our simulation results. The cache size ranges from 2K to 8K blocks. It takes 9 (or 11) bits to index entries of the LR-cache with 2K (or 8K) blocks, since the degree of set associativity is 4. The indexing bits used start from position 23 leftward of the IPv4 destination address (whose rightmost bit is at position 31) of a packet. Given the LR-cache with 2K blocks, for example, the indexing bits are of bit 15 to bit 23 of destination addresses.

A *victim cache* is for keeping blocks which are evicted from a cache due to conflict misses. It is well known that a victim cache can be added to a cache to improve its performance, in particular if the degree of set associativity is small [21], as is the case of LR-caches. A victim cache is a small fully associative cache, aiming to hold those blocks which get replaced so that they are not lost. Entries in the victim cache follow a conventional replacement mechanism (e.g., LRU, FIFO, and random). When a packet is checked against an LR-cache, its corresponding victim cache is also examined simultaneously. A hit, if any, happens to either the cache itself or its corresponding victim cache, but not both. The victim cache usually contains a small number of entries. In this study, each LR-cache is equipped with a victim cache of eight blocks, which are found to be adequate for effective lookup performance improvement by avoiding most conflict misses.

3.3 Overall Lookup Flows

To explain overall lookup flows, let us consider a packet arrival immediately after a table update (when all LR-cache entries are in the invalid state). The packet terminates at one LC (referred to as the *arrival LC*, denoted by LC_{ar}), where the packet header is extracted and delivered to the LR-cache in LC_{ar}. A cache miss will result, and a cache entry is created for the lookup result of this packet header (referred to as the packet hereinafter). Bits of appropriate positions in the destination address of the packet are then examined (by LR1, an LR detector composed of "XOR" logics, shown in Fig. 3) to decide if the packet lookup is to be done locally or remotely, and the bit positions used for examination are those selected for table partitioning (mentioned above). If a local lookup is to be carried out, the packet is moved to the FE of LC_{ar} for longest-prefix matching (based on any software matching algorithm devised earlier for this purpose, and a more efficient algorithm yields a smaller time penalty when there is no hit in the LR-cache). After the lookup result is obtained, it is sent to the LR-cache to complete the corresponding block, with its M bit set to LOC. This cached result will satisfy later lookup requests for an identical destination address much faster, no matter whether those requests are originated from LC_{ar} or other LCs.

On the other hand, if a remote lookup is decided (by detector LR1), the packet is moved to the Outgoing Queue (see Fig. 3) ready for delivery over the switching fabric to its home LC, denoted by LC_{ho}, where the lookup flow then follows as if the packet had arrived there. Specifically, after received by LC_{ho} and put in its Input Queue, the packet is first searched over the (on-chip) LR-cache therein. If a cache miss results, a block is reserved to hold the lookup result of this packet. The packet is then forwarded (through LR1) to the FE of LC_{ho} for longest-prefix matching, whose result later finishes the reserved block, with its M bit set to LOC. A reply is also produced (through LR2 at the upper left corner depicted in Fig. 3) and put in the Outgoing Queue of LC_{ho} for transmission through the fabric back to LC_{ar}. This reply completes the cache block created previously in the LR-cache of LC_{ar}, with the M bit of the block set to REM. The block created in the LR-cache of LC_{ho} thereafter serves

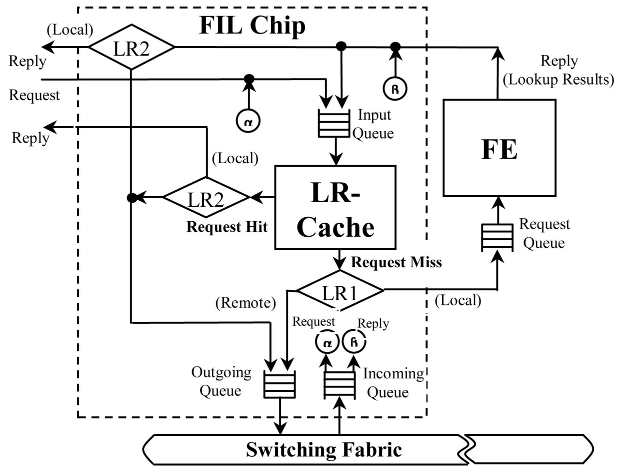


Fig. 3. LR-cache and relevant logics housed inside the FIL chip, where an LR1 (or LR2) detector examines the destination (or originator) address.

to quickly reply upcoming lookup requests (of the same address) originated from *any* LC, whereas the cache block created in LC_{ar} takes care of forthcoming lookup requests from LC_{ar}. While multiple copies of the same lookup result, one in an LC, may be created in the router this way, those copies not only serve multiple packets (from different LCs) with the same destination address concurrently, but also effectively cut down traffic over the switching fabric and the load to the LR-cache in LC_{ho}. As a result, SPAL can yield high lookup performance, partly due to the facts that packets arriving at an LC closely together (in time) will have good chances heading for the same destination, and that parallelism in packet lookups exists.

As blocks with the LOC status and those with the REM status compete in each LR-cache, a proper mix of LOC blocks and REM blocks is necessary. Fortunately, the coverage of IP address space by each LR-cache under SPAL is improved substantially due to table partitioning, which leads to the forwarding table in each LC involving only a small fraction of the prefixes maintained in the routing table. The number of LOC blocks required to achieve a given performance level drops as Ψ (the number of LCs) increases, whereas at the same time, a bigger percentage of lookup requests at each LC will be homed remotely, urging more cache blocks for remote lookup results (i.e., more REM blocks). For a given Ψ, fewer blocks should be allocated for REM blocks when the size of the LR-cache shrinks, in order to get the best lookup performance, since the cost is far smaller over the fabric than longest-prefix matching execution. These phenomena will be uncovered by our simulation outcomes in Section 5.

4 Results of Table Partitioning

Two routing tables were obtained for our evaluation use, one being the FUNET routing table with 41,709 prefixes given in [24] (called RT_1) and the other in AS1221 with 175,853 prefixes obtained on 4 September 2004 [2] (called RT_2). Unlike any existing router, a SPAL-based router calls for partitioning the routing table in accordance with the number of LCs (which can be of any integer, not necessary a power of 2) in the router. The results of table partitioning are exemplified by RT_1 and RT_2.

Following the two criteria for table partitioning stated in Section 3.1, we arrived at the two desired bit positions for fragmenting RT_1 (or RT_2) into four partitions: bits 17 and 18 (or 14 and 25), giving rise to subset sizes of 10,465, 10,523, 10,493, 10,889 (or 51,095, 50,886, 51,029, 49,977). If RT_1 (or RT_2) is to be partitioned into 16 fragments, the preferred partitioning bit positions are found to be 16, 18, 19, 20 (or 17, 18, 20, 23), producing subset sizes of 2,594, 2,851, 2,814, 2,899, 2,494, 2,831, 2,813, 2,517, 2,852, 2,976, 2,912, 2,899, 2,745, 2,523, 2,994, 2,425 (or 16,258, 16,420, 17,495, 18,879, 16,676, 16,240, 17,023, 15,176, 13,431, 16,973, 19,281, 14,959, 14,667, 14,145, 13,835, 14,058).

4.1 Storage Sizes

Three distinct tries (namely, the DP trie [12], the Lulea trie [11], and the LC trie [24]) have been implemented to assess the storage sizes needed for all ROT-partitions after fragmenting RT_1 and RT_2. Under the DP trie for RT_1 (or RT_2) with $\Psi = 4$, the storage sizes of the four tries built and held in LCs after partitioning are 209, 216, 217, and 220 Kbytes (or 917, 914, 917, and 898 Kbytes), respectively, assuming that each node in the trie consists of one byte for the index field plus four bytes for each of the five pointers. Since the trie size before partitioning is 859 (or 3,441) Kbytes, the DP trie sees the amount of SRAM reduction in each LC caused by partitioning to exceed 638 (or 2,524) Kbytes under RT_1 (or RT_2) with $\Psi = 4$. The DP trie after partitioning for $\Psi = 16$ reduces its size down to the range of 50 and 62 Kbytes (or of 165 and 288 Kbytes) with respect to RT_1 (or RT_2). As a result, storage reduction in each LC due to partitioning is no less than 795 (or 3,153) Kbytes under RT_1 (or RT_2) when Ψ equals 16. Note that the reduction amount will be much larger under IPv6. Total SRAM amounts required for the DP trie after (or without) partitioning are depicted in Fig. 4 under DP_S (or DP_W).

For the Lulea trie [11] (whose storage requirement is often the lowest) with $\Psi = 4$ under RT_1 (or RT_2), the partitioned tables in four LCs require 90, 91, 89, and 87 Kbytes (or 504, 494, 505, and 491 Kbytes), respectively, as opposed to roughly 260 (or 978) Kbytes in an LC of a conventional router without partitioning. This indicates an SRAM reduction in each LC caused by partitioning for $\Psi = 4$ under the Lulea trie to be 169 (or 473) Kbytes or beyond, when RT_1 (or RT_2) is concerned. For $\Psi = 16$, the Lulea trie sees its size to be no more than 39 (or 163) Kbytes in any LC after partitioning RT_1 (or RT_2), giving rise to a savings of at least 221 (or 815) Kbytes in each LC. Total SRAM amounts required for the Lulea trie after (or without) partitioning are depicted under LL_S (or LL_W) in Fig. 4. Similarly, the LC-trie [24] under RT_1 (or RT_2) enjoys storage reduction in any LC due to SPAL by no less than 815 (or 1,961) Kbytes, for $\Psi = 4$ with a fill factor of 0.25. The storage saving amount in an LC increases to over 1,025 (or 2,456) Kbytes for RT_1 (or RT_2) under $\Psi = 16$ with the same fill factor (of 0.25). Therefore, SPAL always leads to a far bigger SRAM savings in each LC than the size of the LR-cache incorporated therein (i.e., 24 Kbytes) under the three distinct tries we have implemented and examined.

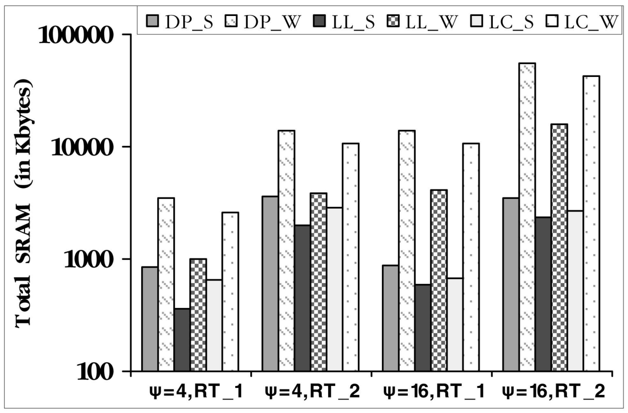


Fig. 4. Total SRAM (in Kbytes) required for different tries under IPv4.

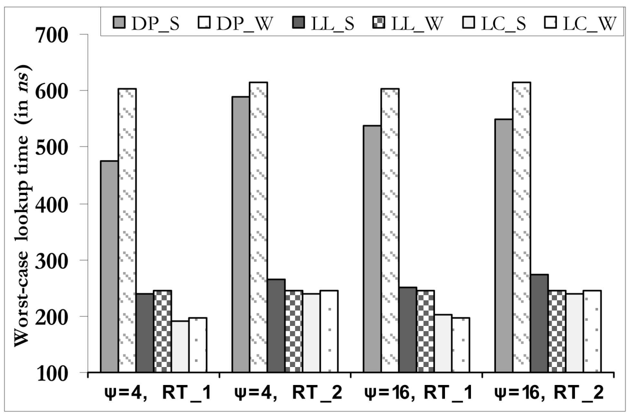


Fig. 5. Worst-case lookup time (in *ns*) under cycle time = 5 *ns* and SRAM access time = 12 *ns*.

4.2 Worst-Case Lookup Times

The time taken for a software lookup involves two components, one due to multiple memory accesses and the other due to program execution (involving some one hundred instructions or so). The former time component dictates the worst-case lookup time for a given longest-prefix matching algorithm employed in the FE of an LC, while the latter one is machine-dependent but typically is no less than 100 *ns*, irrespective of whether a lookup is the worst case or not. The worst-case lookup time for an existing router is thus proportional to the number of memory accesses, each of which takes, say, 12 *ns* when prefixes are held in off-chip SRAM (e.g., the L3 data cache, if existing; otherwise, the L2 data cache, of the FE processor). The numbers of worst-case memory accesses under different tries have been obtained via our actual implementations.

In this work, the latency over the switching fabric is assumed to be size-dependent and is measured in terms of the *cycle time*, which signifies the duration of a visit to an LR-cache and equals 5 *ns*. (Note that this cycle time of 5 *ns* is not impossible, since on-chip SRAM is used for LR-caches and the access time of such SRAM can be as low as 1 *ns*, if its size is small, as is the case of our LR-caches.) The latency over the switching fabric for $\Psi = 4$ (or 16) is assumed to be one cycle (or two cycles). This latency assumption is conservative when compared with a previous switch cache design [20], which is built by 4×4 switching elements incorporated with caches operating at 200 MHz (like Cavallino [5]). Without any cache, our switching fabric sized 4 is simpler than a 4×4 switch cache element and can surely operate at 200 MHz.

When compared with its counterpart, a SPAL-based router with $\Psi = 4$ (or 16) sees the worst-case lookup time to increase by: a round-trip latency over the fabric equal to two (or four) cycles plus two cycles for visiting two LR-caches, one at LC_{ar} and the other at LC_{ho}, amounting to 20 (or 30) *ns* extra in total. However, the matching algorithm executed by an FE under SPAL can be shorter in the worst case due to fewer memory accesses, as a result of partitioning the routing table. When the DP trie [12] is considered with respect to RT_1 (or RT_2) under $\Psi = 4$, for example, our implementation results indicate that in the worst case, search through the four partitioned tries involves 38, 34, 38, 38 (or 39, 39, 35, 39) node accesses, respectively, in comparison with 42 (or 43) accesses to the DP trie without partitioning. As a result, the DP trie enjoys a reduction of no less than four node accesses under SPAL with $\Psi = 4$ when RT_1 (or RT_2) is considered; this reduction translates to shortening the matching search time by at least 48 *ns* in the worst case, when a node access is assumed optimistically to involve just one SRAM access of 12 ns (since off-chip SRAM typically exhibits access time ranging from 8 to 15 *ns* and a node in the DP trie contains some $1 + 4 \times 5 = 21$ bytes [12]). This search time reduction well offsets the extra time incurred under SPAL. Likewise, SPAL for $\Psi = 16$ under RT_1 (or RT_2) gives rise to no more than 34 (or 35) node accesses over any partitioned trie in the worst case according to our actual implementations, yielding a search time savings of at least 96 *ns*. The worst-case lookup times in a SPAL-based (or commercial) router following the DP trie are illustrated under DP_S (or DP_W) in Fig. 5, where the worst-case lookup times are shortened as a result of SPAL.

If the Lulea trie is adopted, a far smaller forwarding table in each FE after partitioning may avoid the dense chunks and the very dense chunks in the third level or even avoid all the third level chunks of the trie [11], reducing tow or even four memory accesses (from $4 + 4 + 4$ down to $4 + 4 + 2$ or even to $4 + 4$) for the worst case; this memory access reduction translates to a savings of 24 *ns* or even 48 *ns* (given the SRAM access time of 12 *ns*). Under RT_1, any Lulea trie for both $\Psi = 4$ and $\Psi = 16$ after partitioning sees the worst case search to involve 10 memory accesses, in contrast to 12 memory accesses when the routing table is not partitioned, according to real implementations. The Lulea trie thus experiences search time reduction by two memory accesses in the worst case. The worst-case lookup times in a SPAL-based (or commercial) router following the Lulea trie are shown under LL_S (or LL_W) in Fig. 5. Similarly, if the LC-trie is employed, a smaller forwarding table under SPAL usually reduces the maximum path length in the trie constructed. Considering RT_1 (or RT_2) under a fill factor of 0.25, for example, the search path depth in the LC-trie after partitioning for $\Psi = 4$ is bounded by 6 (or 10), whereas the maximum trie depth without partitioning equals $5 + 3 = 8$ (or $6 + 6 = 12$) according to the implementation provided in [24], yielding a savings of at least two memory accesses in the worst case; this amounts to a reduction of 24 *ns*. Likewise, the LC-trie with $\Psi = 16$ under RT_1 (or RT_2) sees its maximum trie depth after partitioning to be no more than six (or nine), exhibiting the worst case search time savings of no less than two (or three) memory accesses. This is likely to yield a smaller worst-case lookup time, as depicted in Fig. 5 (denoted by LC_S).

5 Performance Evaluation

Trace-driving simulation was employed to evaluate the performance of SPAL-based routers under different LR-cache sizes and Ψ values. This includes simulation scenarios of different LC speeds and various longest prefix matching algorithms under many traces available to the public. Note that the LC speeds considered are those after link aggregations, if needed; for example, Cisco's 12000 Series routers allow multiple links of varying speeds (terminating at separate ports of an LC) to be aggregated in each LC for up to 10 Gbps.

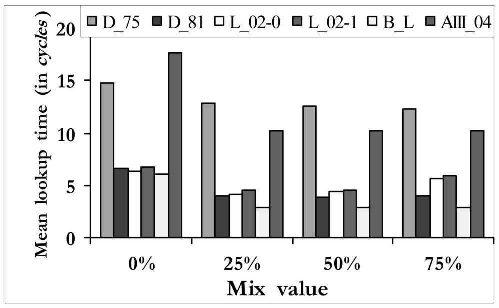
5.1 Simulation Methodology

Our simulator takes as its input, the packet streams fed to all LCs and the mean longest prefix matching time per lookup in FEs. The packet streams were derived from various traces of actual packet destinations collected and posted [28], [39], one stream for each LC. For Abilene-I, Abilene-II, Bell Labs-I, Abilene-III, and other data sets in the PMA long traces archive (details about where, when, and how traces were collected and about the trace format can be found in [28]), the destinations of IP packet records (each consisting of 64 bytes) in the traces were employed as packet streams to drive our simulation studies. For the WorldCup98 data set (which contained all request logs from 30 April to 26 July 1998 with more than 1.35 billion requests in total [39]), the clientID field of each request (i.e., a mapped IP address of the request originator or proxy) was employed to drive our simulator. Two different LC speeds were evaluated: 10 Gbps and 40 Gbps. Given a simulated LC speed of 10 (or 40) Gbps, packets of varying length are generated in a way that on an average, they together amount to the given speed, with the mean packet length assuming to be 256 bytes and the smallest packet size equal to 40 bytes [36]. Under the clock cycle of 5 *ns* simulated for the LC speed of 40 (or 10) Gbps, one packet was generated in anywhere from two cycles to 18 cycles (or from six cycles to 74 cycles). Given a trace, once a packet is generated at an LC, its destination was supplied by the trace. This approach enables one trace of addresses to feed different numbers of LCs whose packet generation processes can be specified individually and are dictated by their respective LC speeds under consideration. Meanwhile, traffic locality of the trace is reduced in this way, giving rise to pessimistic (i.e., conservative) simulation results. Each LC in our simulation produces 300,000 packets, which correspond to a time period of roughly 15 (or 60) *ms* for the mean packet length of 256 bytes under the LC speed of 40 (or 10) Gbps. This duration is so chosen because prefix changes occur some 20 times on an average [3] and possibly up to 100 times [32] per second, and every prefix change leads to the cache contents in LR-caches being flushed entirely.

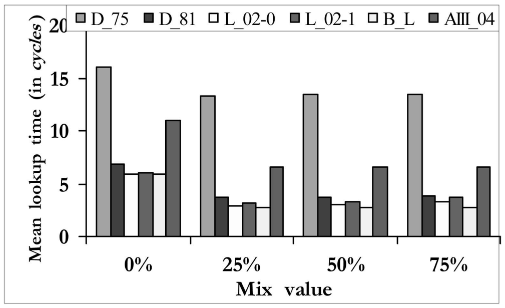
The LR-cache organization can be specified by its size (in terms of blocks), given that its degree of set associativity equals 4, the victim cache size is 8, and each cache block is set to hold only *one* lookup result. When a conventional replacement policy is needed, the LRU is applied. Likewise, the replacement policy in the victim cache follows the LRU. In a cycle (of 5 *ns*), at most one packet is checked against an LR-cache (see Fig. 3); if the check leads to a miss, the cache is then updated accordingly. Each table lookup consists of multiple memory accesses and the execution of the software code which realizes longest prefix matching. The mean number of memory accesses varies widely from one matching algorithm to another. We have implemented various tries to measure the average numbers of memory accesses per lookup under the two sets of prefixes utilized for this study, namely, RT_1 and RT_2. It is found that the Lulea trie [11] requires 6.2 (or 6.6) memory accesses per lookup on an average for RT_1 (or RT_2), while the DP trie [12] yields about 16 memory accesses per lookup for either set of prefixes. Since the trie is kept in off-chip SRAM (e.g., the L3 data cache, if existing; otherwise, the L2 data cache), the memory access time is assumed to be 12 *ns* and the code execution time is 120 *ns* (for executing some 100 instructions per lookup) in our simulation. This assumption leads to a matching search time of roughly 40 cycles (of 5 *ns* each) in FEs under the Lulea trie and of 62 cycles or so under the DP trie. SPAL was evaluated under these search times in FEs.

5.2 Outcomes and Discussion

Extensive simulation results for RT_1 and RT_2 were gathered and found to exhibit a similar trend; therefore, only the results for RT_2 are presented here. They confirm that typical packet streams indeed **have sufficient temporal locality** to make the LR-cache effective, according to traces collected in 1998, 2002, and 2004 available to the public [28], [39]. These defy an earlier projection based on the trace data gathered in 1987 and 1995, that access locality in packet streams would decrease over time [38], and are in agreement with another study according to real traces collected by the authors in 1998 over the only external link at T3 rate for connecting the Brookhaven National Laboratory [7], [8]. The results are for different cases: 10 Gbps and 40-cycle lookup, 10 Gbps and 62-cycle lookup, 40 Gbps and 40-cycle lookup, and 40 Gbps and 62-cycle lookup. Since those cases see their results follow a similar trend, in this paper, we present the simulation outcomes only for *the case of 40 Gbps and 40-cycle lookup*, under two traces from WorldCup98, namely, D_75 (for 9 July 1998) and D_81 (for 15 July 1998), two traces from the Abilene-I data set (on 14 August 2002) in the PMA Long Traces Archive, namely, L_02-0 and L_02-1, and the Bell Labs-I trace (on 19 May 2002) from the same archive. In addition, the Abilene-III trace (comprising 4-hour real Internet backbone traffic over an OC192c link between the Indianapolis router node and Kansas City on 1 June 2004 [28]) in the same archive, denoted by AIII_04, is also employed. Note that many other traces in [28], [39] were examined by our extensive simulation studies and their results all fall within or around the ranges signified by these six traces. Our performance measures of interest include the average lookup time, the load balancing gauge, and the normalized transient (defined below).

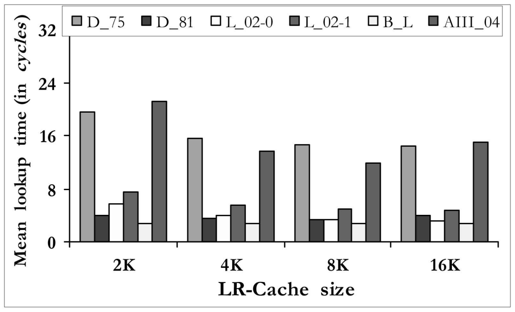


(a)

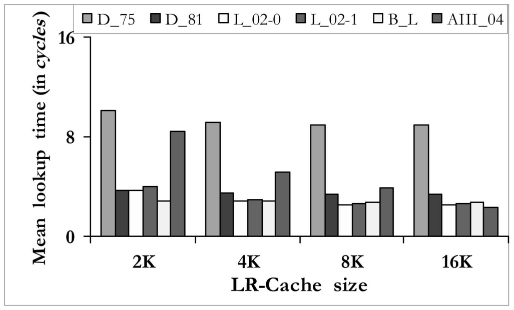


(b)

Fig. 6. Mean lookup time (in cycles) versus mix value (γ) for $\Psi = 4$. (a) $\beta = 2K$ and (b) $\beta = 4K$.



(a)



(b)

Fig. 7. Mean lookup time (in cycles) versus LR-cache size (β). (a) $\Psi = 3$ and (b) $\Psi = 16$.

As the set of associativity degree is chosen to be 4 and blocks in a set can be employed to hold LOC and REM lookup results, we examined how the "mix" value (γ, reflecting percent of blocks devoted for REM results) affected lookup performance under various LR-cache sizes (β, in blocks, with a block holding one lookup result) for Ψ ranging from 1 to 64. The simulation outcomes all follow a trend similar to that signified in Fig. 6, where the mean lookup time (in cycles of 5 *ns* each) as a function of γ under the six traces is depicted for $\Psi = 4$ with β equal to 2K and 4K. This figure reveals that $\gamma = 50$ percent typically yields best (or nearly best) performance, namely, two cache blocks per set are for REM results, because 1) sufficient blocks are always needed to cache local lookup results and 2) lookup execution is far more expensive than the round-trip latency over the switching fabric. Hence, we present subsequently the outcomes only for $\gamma = 50$ percent.

We investigated the impact of the LR-cache size on SPAL performance, with simulation results under the six traces demonstrated in Fig. 7 and Fig. 8. The average lookup time (in cycles) as a function of the cache size (β) for $\Psi = 3$ and 16 is depicted in Fig. 7, where a table lookup carried out at an FE (after a miss in the LR-cache) takes 40 cycles. For any given trace, a larger β consistently yields a shorter lookup time until it reaches a saturation point (determined largely by Ψ); with $\beta = 4K$, the mean lookup times under SPAL sized $\Psi = 3$ (or 16) drop below 15.6 (or 9.2) cycles for all the traces shown, translating to a lookup speed beyond 12.8 (or 21.7) million packets per second for each LC. Therefore, a SPAL-based router sized 3 (or 16) can forward more than 38 (or 347) million packets per second, provided that $\beta \geq 4K$. In contrast, a current router without table partitioning nor LR-caches experiences the mean lookup time equal to 200 *ns* (i.e., 40 cycles) if the queuing time of the FE is ignored optimistically; that is equivalent to 5 million lookup per second per LC. Thus, a SPAL-based router with $\Psi = 3$ (or 16) accelerates packet lookups by 2.5 (or 4.3) times, when compared with its commercial counterpart, despite its reduced total SRAM amount and a possibly shorter worst-case lookup time. If β grows beyond the saturation point (say, 8K under $\Psi = 3$ for all traces), mean lookup performance stays roughly unchanged or even drops slightly.

Mean lookup performance versus Ψ (i.e., the number of LCs) under $\beta = 4K$ and $\gamma = 50$ percent is illustrated in Fig. 8. According to the figure, a larger Ψ generally leads to a lower

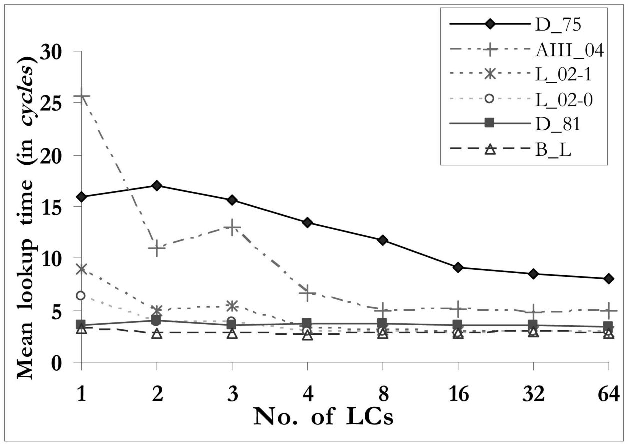


Fig. 8. Mean lookup time (in cycles) versus Ψ under β = 4K and γ = 50 percent.

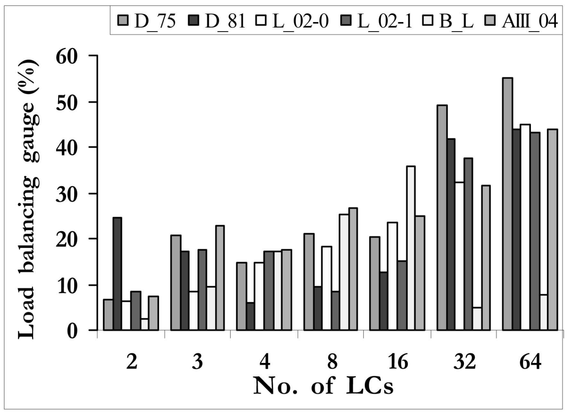


Fig. 9. Load balancing gauge ($\triangle_{\mathrm{FE}}$) versus Ψ under β = 4K and γ = 50 percent.

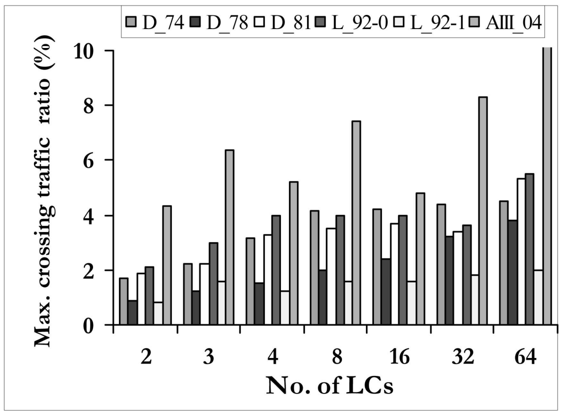


Fig. 10. Maximal crossing traffic ratio (percent) versus Ψ under β = 4K and γ = 50 percent.

mean lookup time for any trace, because of better address space coverage in each LC (due to fewer prefixes in its forwarding table) and increased parallelism offered by more FEs (for longest-prefix matching execution). Specifically, the mean lookup time for trace L_02-0 drops from more than six cycles down to less than three cycles, if Ψ rises from 1 to 16, translating to a speedup factor of more than 2 as a result of finer routing table fragmentation (and, thus, partitioning the IP packet streams into more subsets) for larger parallelism. When the LR-cache is incorporated in each LC while the routing table is not partitioned (as treated in an earlier processor caching work [8], [16]), the mean lookup time will be independent of Ψ and be always equal to that of $\Psi = 1$ depicted in Fig. 8. The benefits due to incorporating LR-caches in LCs then drop substantially because 1) the LR-cache has a larger coverage of the address space (i.e., the whole routing table, instead of a small fraction of it, like SPAL) and 2) the lookups of same IP addresses have to be repeated in different LCs, discounting the purpose of caches. Note that the number of LCs can be of any integer, not limited to powers of 2 as mentioned earlier.

The partitioning bits chosen under SPAL are simply based on prefixes in the routing table without regard to IP packet streams. The bits so chosen aim to minimize and equalize the forwarding table size at each LC, irrespective of traffic whose distribution of its packet destinations change dynamically. It is interesting to observe that the partitioning bits obtained this way often give rise to reasonably balanced load at each FE for the range of Ψ examined, ensuring that concurrent packet lookups are done rather evenly across all LCs to achieve high performance. Let $\triangle_{\mathrm{FE}}$ denote a *load balancing gauge* of FEs, expressed by $(L_{\mathrm{H}} - L_{\mathrm{L}})/L_{\mathrm{H}}$, where L_{H} (or L_{L}) is the highest (or lowest) load among all FEs in the router, with the load of an FE measured by the total number of lookups performed in the FE. Load balancing gauge ($\triangle_{\mathrm{FE}}$) versus Ψ under the cache size of β = 4K and the mix value of γ = 50 percent is shown in Fig. 9, where a lower $\triangle_{\mathrm{FE}}$ signifies a better balanced outcome. As can be found in the figure, all traces result in fairly balanced load (with $\triangle_{\mathrm{FE}} <$ 56 percent always) over the Ψ range evaluated. While our partitioning approach takes no specific traffic characteristics into consideration, it often makes all FEs carry out table lookups concurrently with reasonably balanced load, yielding a small mean lookup time.

The amount of crossing traffic over the switching fabric for looking up routing tables remotely can be reflected by the fraction of total lookup requests which are originated from remote LCs and handled by a given LC, called the crossing traffic ratio of the LC (denoted by θ). The crossing traffic ratio varies from one LC to another, and the maximal θ (represented by $\theta_{\max}$) versus Ψ is depicted in Fig. 10, where β is 4K and γ equals 50 percent. As expected, $\theta_{\max}$ grows in general when Ψ rises, but it is no more than 5.5 percent for Ψ up to 64, except for the AIII_04 trace (whose $\theta_{\max}$ approaches 10.2 percent for $\Psi = 64$). Crossing traffic under SPAL therefore accounts for a very small fraction of total traffic over the switching fabric.

As explained earlier in Fig. 6, the LR-cache at each LC holds both LOC and REM lookup results. It is interesting to know how frequent those LOC lookup results at LCs are employed to satisfy remote lookup requests issued from other LCs. To this end, cached remote entries utilization is calculated via dividing the number of remote "hit" requests by the number of remote requests served by FEs, and the utilization results as a function of Ψ under β = 4K and γ = 50 percent is demonstrated in Fig. 11. As can be observed from the figure, cached LOC lookup results benefit a large amount of subsequent remote requests for all traces but B_L under $\Psi \geq 4$. Without caching LOC lookup results, the load of table lookups at every FE will grow considerably.

Since SPAL flushes all the cache contents upon its operation or after each batch of routing table updates, it is thus insightful to gauge the period for cache initialization, called the warm-up period (in cycles), defined as the number of cycles taken until the cache hit rate reaches a specified level, say 75 percent. Each LC is assumed to process 300,000 packets before a batch of table updates occurs, followed by flushing out all LR-cache contents. The *normalized transient* is designated as the warm-up period divided by the total service time (in cycles) for every LC to finish its 300,000 packets.

Normalized transient versus the number of LCs (Ψ) for β = 4K and γ = 50 percent under different traces is illustrated in Fig. 12, where a smaller normalized transient reflects better access locality and therefore a lower average lookup time. It can be observed that the normalized transient is trace-dependent and is usually very small, ranging from 0.03 percent to less than 1.3 percent, for $\Psi \geq 2$. When Ψ equals 1, the

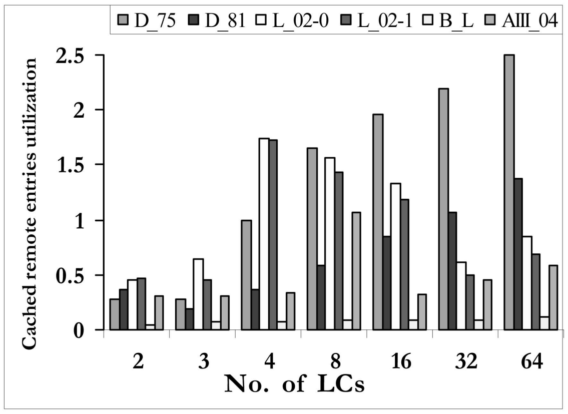


Fig. 11. Cached remote entries utilization versus Ψ under β = 4K and γ = 50 percent.

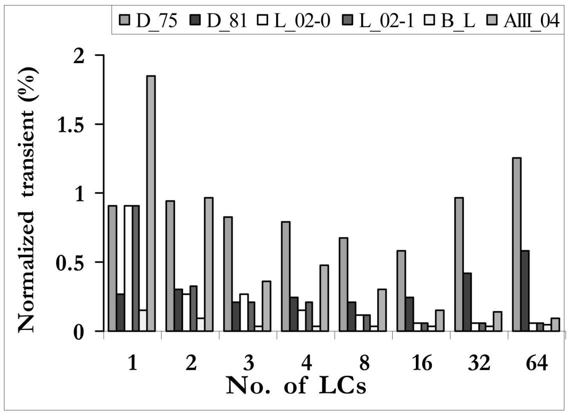


Fig. 12. Normalized transient versus Ψ under β = 4K and γ = 50 percent.

normalized transient reaches 1.85 percent for Trace AIII_04. For a given trace, the normalized transient drops when $\Psi(\geq 2)$ increases, as expected, since more LCs are then involved in packet lookups concurrently to populate LR-caches with lookup results sooner. The outcomes indicate that SPAL is effectively suitable for the case of batch updates to the routing table, given its tiny normalized transients. While not demonstrated here, our simulation results also reveal that the normalized transient is rather insensitive to the cache size (β) for every trace examined, because SPAL would hold enough IP lookup results to reach the specified hit rate (of 75 percent) after a certain number of cycles from the time its caches were flushed, as long as β is greater than or equal to 2K. Even with a simple flushing mechanism for the LR-cache contents upon a routing table update, the SPAL-based router is observed to exhibit speedy forwarding quickly after each update because every lookup result from an FE is held not only in the LR-cache of the home LC, but also in each remote LR-cache where packets with the same destination arrive, as stated in Section 3.3.

6 Conclusion

A speedy packet lookup (SPAL) technique has been investigated for scalable high-performance routers, realized by fragmenting the BGP routing tables and incorporating a small cache (say $4K \times 6$ bytes under IPv4, called the LR-cache) for keeping lookup results. The forwarding table housed in each LC (linecard) includes only a small subset of prefixes (in the routing table), leading to a significant drop in the SRAM requirement and a possibly improved worst-case lookup time. The number of LCs (i.e., fragments of a BGP table) can be an arbitrary integer, not necessarily a power of 2. The LR-cache enables quick replies to subsequent requests (for looking up same addresses) originated from the home LC as well as other LCs, greatly enhancing mean lookup performance of a SPAL-based router. Trace-driven simulation is adopted to assess the performance measures of interest, and the simulation outcomes under various traces demonstrate that a SPAL-based router delivers much faster mean lookups than any existing router whose forwarding tables are all identical and have the same number of prefixes as the core routing table. For a given LR-cache size and any trace, mean lookup performance typically improves for a larger Ψ, as a result of finer routing table fragmentation so that each LR-cache in an LC then achieves a better address space coverage. While the partitioning bits are chosen according to the prefixes in a given routing table and irrespective of packet streams, it is found that the partitioning bits obtained this way often yield reasonably balanced load across FEs, ensuring good scalability (with respect to the growth of routing tables) and high lookup concurrency to arrive at a small mean lookup time. In addition, SPAL is equally applicable to IPv6, unlike other known software or hardware-based forwarding enhancement approaches which are typically far more expensive, if possible, for IPv6. Capable of speeding up packet lookups while drastically lowering SRAM amounts needed, SPAL is ideally suitable for the new generation of scalable high-performance routers.

Acknowledgments

The author thanks G. Wang and R. Dugyala for their participation in discussions and their efforts in helping to develop and rectify the simulator, collecting simulation results, and drawing figures. This work was supported in part by the US National Science Foundation under Grants EIA-9871315 and CCR-0105529. Its preliminary version was delivered at the 2004 International Conference on Parallel Processing, August 2004.

References

[1] M. Akhbarizadeh and M. Nourani, "An IP Packet Forwarding Technique Based on Partitioned Lookup Table," *Proc. 2002 IEEE Int'l Conf. Comm.*, Apr./May 2002.
[2] AS1221 BGP Table Data, http://bgp.potaroo.net/as1221/bgp-active.html, routing table snapshot taken during Sept. 4-6, 2004.
[3] A. Basu and G. Narlikar, "Fast Incremental Updates for Pipelined Forwarding Engines," *Proc. IEEE Conf. Computer Comm. (INFOCOM '03),* Apr. 2003.
[4] BGP Table Data, http://bgp.potaroo.net, Sept. 2004.
[5] J. Carbonaro and F. Verhoorn, "Cavallino: The Teraflops Router and NIC," *Proc. Fourth Symp. High-Performance Interconnects (Hot Interconnects 4),* Aug. 1996.
[6] H. Chan, H. Alnuweiri, and V. Leung, "A Framework for Optimizing the Cost and Performance of Next-Generation IP Routers," *IEEE J. Selected Areas in Comm.*, vol. 17, pp. 1013-1029, June 1999.
[7] T. Chiueh and P. Pradhan, "High-Performance IP Routing Table Lookup using CPU Caching," *Proc. IEEE Conf. Computer Comm. (INFOCOM '99),* pp. 1421-1428, Apr. 1999.

[8] T. Chiueh and P. Pradhan, "Cache Memory Design for Internet Processors," *IEEE Micro,* vol. 20, Jan./Feb. 2000.

[9] Cypress Semiconductor Corp., "Network Search Engines,"http://www.cypress.com/, Dec. 2003.

[10] Cisco Systems, *Cisco 12016 Gigabit Switch Router, Data Sheet,* http://www.cisco.com, 2001.

[11] M. Degermark et al., "Small Forwarding Tables for Fast Routing Lookups," *Proc. ACM SIGCOMM 1997 Conf.,* pp. 3-14, Sept. 1997.

[12] W. Doeringer, G. Karjoth, and M. Nassehi, "Routing on Longest-Matching Prefixes," *IEEE/ACM Trans. Networking,* vol. 4, no. 1, pp. 86-97, Feb. 1996.

[13] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting," *Proc. ACM SIGCOMM 2002 Conf.,* pp. 323-336, Aug. 2002.

[14] W. Fang and L. Peterson, "Inter-AS Traffic Patterns and Their Implications," *Proc. 1999 IEEE Global Internet Symp.,* Dec. 1999.

[15] M. Galles, "Spider: A High-Speed Network Interconnect," *IEEE Micro,* vol. 17, pp. 34-39, Jan./Feb. 1997.

[16] K. Gopalan and T. Chiueh, "Improving Route Lookup Performance Using Network Processor Cache," *Proc. Supercomputing Conf. 2002,* Nov. 2002.

[17] P. Gupta, S. Lin, and N. McKeown, "Routing Lookups in Hardware at Memory Access Speeds," *Proc. IEEE Conf. Computer Comm. (INFOCOM '98),* pp. 1240-1247, Apr. 1998.

[18] Hitachi, Ltd., "The Hitachi GR2000 Gigabit Router Series," http://www.internetworking.hitachi.com, 2002.

[19] G. Huston, "Analyzing the Internet's BGP Routing Table," *The Internet Protocol J.,* vol. 4, no. 1, Mar. 2001.

[20] R. Iyer and L. Bhuyan, "Switch Cache: A Framework for Improving the Remote Memory Access Latency of CC-NUMA Multiprocessors," *Proc. Fifth Int'l Symp. High-Performance Computer Architecture,* pp. 152-160, Jan. 1999.

[21] N. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," *Proc. 17th Ann. Int'l Symp. Computer Architecture,* pp. 364-373, May 1990.

[22] Juniper Networks, Inc., "T-Series Routing Platforms: System and Packet Forwarding Architecture," white paper, http://www.juniper.net, Apr. 2002.

[23] NetLogic Microsystems, "Network Search Engines (NSEs)," http://www.netlogicmicro.com/products/products.html, Jan. 2004.

[24] S. Nilsson and G. Karlsson, "IP-Address Lookup Using LC-Tries," *IEEE J. Selected Areas in Comm.,* vol. 17, no. 6, pp. 1083-1092, June 1999.

[25] D. Pao et al., "Efficient Hardware Architecture for Fast IP Address Lookup," *Proc. IEEE Conf. Computer Comm. (INFOCOM '02),* June 2002.

[26] C. Partridge, "Locality and Route Caches," *Proc. US Nat'l Science Foundation Workshop Internet Statistics and Metrics Analysis,* http://www.caida.org/outreach/isma/9602/positions/partridge.html, Feb. 1996.

[27] Pericom Semiconductor Corp., "Throughput Expansion with FET-Based Crossbar Switching," http://www.pericom.com/, Nov. 2001.

[28] PMA Long Traces Archive, http://pma.nlanr.net/Traces/long/, Passive Measurement and Analysis, Nat'l Laboratory for Applied Network Research, Sept. 2004.

[29] The RACE Multicomputer, vol. 1, version 1.3, Mercury Computer Systems, Inc., 1995.

[30] V. Ravikumar and R. Mahapatra, "TCAM Architecture for IP Lookup Using Prefix Properties," *IEEE Micro,* vol. 24, no. 2, pp. 60-69, Mar./Apr. 2004.

[31] V. Ravikumar, R. Mahapatra, and L.N. Bhuyan, "EaseCAM: An Energy and Storage Efficient TCAM-Based Router Architecture," *IEEE Trans. Computers,* to appear.

[32] M. Ruiz-Sanchez, E. Biersack, and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms," *IEEE Network,* vol. 15, pp. 8-23, Mar./Apr. 2001.

[33] S. Sahni and K. Kim, "An $O(\log n)$ Dynamic Router-Table Design," *IEEE Trans. Computers,* vol. 53, no. 3, pp. 351-363, Mar. 2004.

[34] K. Sklower, "A Tree-Based Packet Routing Table for Berkeley Unix," *Proc. 1991 Winter Usenix Conf.,* pp. 93-99, 1991.

[35] V. Srinivasan and G. Varghese, "Fast Address Lookups Using Controlled Prefix Expansion," *Proc. ACM Sigmetrics '98,* pp. 1-11, June 1998.

[36] K. Thompson, G. Miller, and R. Wilder, "Wide-Area Internet Traffic Patterns and Characteristics," *IEEE Network,* vol. 11, pp. 10-23, Nov./Dec. 1997.

[37] N. Tzeng, "Hardware-Assisted Design for Fast Packet Forwarding in Parallel Routers," *Proc. 2003 Int'l Conf. Parallel Processing,* pp. 11-18, Oct. 2003.

[38] M. Waldvogel et al., "Scalable High-Speed Prefix Matching," *ACM Trans. Computer Systems,* vol. 19, no. 4, pp. 440-482, Nov. 2001.

[39] WorldCup98 Data Set, http://ita.ee.lbl.gov/html/contrib/WorldCup.html, The Internet Traffic Archive, Lawrence Berkeley Nat'l Laboratory, Apr. 2000.

[40] Xelerated, Inc., "Co-Processors," http://www.xelerated.com/templates/page.aspx?page_id=197, Oct. 2003.

[41] R. Zakon, "Hobbes' Internet Timeline v6.0," 2003, http://www.zakon.org/robert/internet/timeline.

[42] K. Zheng et al., "An Ultra High Throughput and Power Efficient TCAM-Based IP Lookup Engine," *Proc. IEEE Conf. Computer Comm. (INFOCOM '04),* Mar. 2004.

Nian-Feng Tzeng received the PhD degree in computer science from the University of Illinois at Urbana-Champaign in 1986. He is currently a professor with the Center for Advanced Computer Studies, University of Louisiana at Lafayette, which he joined in 1987. His current research interests are in the areas of computer communications and networks, grid and peer-to-peer computing, and high-performance and dependable systems. He had been on the editorial board of the *IEEE Transactions on Parallel and Distributed Systems* (1998-2001) and on the editorial board of the *IEEE Transactions on Computers* (1994-1998), and had served as coguest editor of a special issue of the *Journal of Parallel and Distributed Computing* (September 1995), and as a distinguished visitor of the IEEE Computer Society (1994-1997). He was the chair of the technical committee on distributed processing, the IEEE Computer Society, from 1999 to 2002, and the technical program chair of the 10th International Conference on Parallel and Distributed Systems, July 2004, and has been on the technical program committees of various conferences. Dr. Tzeng is a member of the ACM and the recipient of the outstanding paper award of the 10th International Conference on Distributed Computing Systems, May 1990. He received the University Foundation Distinguished Professor Award at University of Louisiana in 1997. He is a senior member of the IEEE and the IEEE Computer Society.