SPAL: A Speedy Packet Lookup Technique for High-Performance Routers

Nian-Feng Tzeng
Center for Advanced Computer Studies
University of Louisiana at Lafayette
Lafayette, Louisiana 70504, U.S.A.
tzeng@cacs.louisiana.edu

Abstract

This work introduces and evaluates a technique for speedy packet lookups, called SPAL, in high-performance routers, realized by fragmenting the BGP routing table into subsets. Such a router contains multiple line cards (LCs), each of which is equipped with a forwarding engine (FE) to perform table lookups locally based on its forwarding table (which is a fragmented subset). The number of table entries in each FE drops as the number of LCs in a 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 matching algorithm. SPAL calls for caching the lookup result of a given IP address at its home LC (denoted by LC_{ho}, using the LR-cache), such that the result can satisfy the lookup requests for the same address from not only LC_{ho} but also other LCs quickly, when the switching fabric for interconnecting LCs has a low latency. Lookup results obtained from remote LCs are also held in the LR-cache of a local LC. Our trace-driven simulation reveals that SPAL indeed leads to substantial improvement in mean lookup performance. SPAL may possibly shorten 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 takes no specific traffic into consideration when selecting the partitioning bits, promising good scalability and a small mean lookup time per packet.

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 two years [4], with certain routing tables now involving more than 120K prefixes (see AS1221, AS4637, AS6447 in [4]). As search in routing and forwarding tables is complex, usually based on *longest prefix matching search* to arrive at the most *specific* search result for a given IP address, prefixes are usually organized as a tree-like structure called a *trie*, with its nodes either corresponding to prefixes or forming paths to prefixes, for effective search. It is highly desirable to fit the trie within static RAM (SRAM) for good search performance, requiring a rather large amount of SRAM for the forwarding engine (FE) at each line card (LC), in the form of an L3 data cache. 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. 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 [7, 8, 12], hardware-based lookup designs [10], and hardware-assisted forwarding lookups [5, 16]. This work deals with a technique for accelerating packet lookups in a high-performance router with multiple LCs, each of which has one FE to carry out table lookups (see Fig. 1).

The latency of a small crossbar switch has fallen considerably, resulting from a steady decline in the switching time of crossbars. Recent crossbars enjoy consistently low latencies, as evidenced by the Pericomís P15X1018 crossbar switch, which features 10 ports (of 18-bit width each) running at 133 MHz [13]. One may expect to see a future crossbar with its switching time down to a few *ns*. Such crossbar switches make it possible to build a multistage-based switching fabric for interconnecting a moderate number of LCs in a router, with packet latency over the fabric being 10 *ns* or less.

The opportunity offered by fast switched 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

This work was support in part by NSF under Grants EIA-9871315 and CCR-0105529.

of requests for address lookups at each FE. Together, a hardware-assisted design for speedy packet lookups, called SPAL, is obtained for high-performance routers, with four salient features. First, SPAL drastically lowers the size of the trie (due to few 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 but also possibly shortens the worst-case lookup time as well.

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. Therefore, 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 LR-cache introduced to hold lookup results) tremendously.

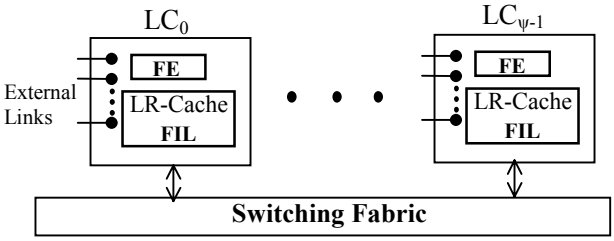

Fig. 1. A SPAL-based router, where FIL refers to fabric interface logics.

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 16 LCs can forward, on an average, over 336 million of packets per second, if each LR-cache involves 4K blocks, when the Lulea trie [7] is adopted for longest prefix matching. This average forwarding ability is 4.2 times faster than that of an existing router, which keeps all prefixes of the routing table in each LC and has no LR-caches. A SPAL-based router may also shorten the worst-case lookup time, in comparison with a conventional router under the same longest prefix matching algorithm.

While IP lookup traffic streams present different characteristics than the data streams of typical computing applications, an independent investigation based on 1998 traces [5, 6] indicated the adequacy of 4K blocks in each cache to reach hit rates higher than 0.93. While the Internet has grown by more than 10 times from Jan. 1996 (with 14 million hosts) to July 2002 (with 162 million hosts), the locality of IP traffic over the Internet **does not drop** based on two separate, independent trace-driven simulation studies using 1998 traces and 2002 traces, perhaps due to the observed fact that a small percentage of flows between AS pairs (say, 9%) in the Internet accounts for a large percentage of total traffic (say, 90%) [9]. 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 [17] and 2002 backbone traffic [14] 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.

2. Background

2.1 Packet Lookups

Packet lookups in routers can be expedited by various approaches, generally classified as software-based or hardware-based ones. 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. An enhanced trie implementation, called a *DP trie* (dynamic prefix trie), was considered to lower the average number of memory accesses upon search [8] and to yield a small code size and low storage requirements. Various algorithms resort to multiple-bit inspection at each search step (rather than single-bit inspection for 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 [15]. The Lulea algorithm constructs a 3-level compressed data structure, with the strides of 16, 8, and 8, respectively, for the first, the second and the third levels [7]. 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) [12]. Search in an LC-trie requires an explicit comparison when arriving at a leaf to ensure a search match. While applicable to 128-bit IPv6 prefixes, software-based approaches 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 multi-bit trie with fixed strides was implemented in hardware to support IP lookups at the speed of memory

accesses [10], with the first level realized by a table with 2^{24} entries addressed by the first 24 IP bits. Each sub-trie in the second level contains 2^8 entries, and the total number of such sub-tries depends on the set of prefixes at hand. The memory requirement of this hardware design is huge (> 32 Mbytes). When a lookup hardware design is applied to 128-bits prefixes, it always takes a massive amount of memory.

2.2 Caching Lookup Results

A network processor equipped with hardware caches for capturing table lookup results has shown to improve overall packet forwarding performance significantly [5, 6]. The caching algorithm described in [5] mapped IP addresses carefully to virtual addresses so as to make use of CPU caches (both L1 and L2) for fast lookups. 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 [6]. It yields better performance when the address range cached in an entry is larger. Two steps of address range merging were considered and a greedy bit selection algorithm was introduced to minimize the total number of address ranges and the size differences among address ranges after mapping [6]. 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 [11, 15], 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, 15], 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. Thus, cache hit rates using the network processor cache after address range merging may not be high usually.

2.3 Partitioning for Parallel Search

A technique has been considered recently [1] for partitioning the routing table into subsets, which can then be searched in parallel. It groups all prefixes of a given length in the routing table as one partition. Clearly, the size of each partitioned subset varies considerably, e.g., the length of size 24 typically accounts for about 50% of all prefixes [2]. Unlike our SPAL, the earlier technique keeps *all* partitioned subsets at each FE for search in parallel [1]; 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.

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 (FILís), and the fabric can be a shared-bus (for a small ψ), a crossbar, or a multistage-based structure, 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 (as will be detailed in Sec. 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 doesnít have to be a power of 2 and can be any integer, say 3, 5, 6, 7, etc. For easy explanation in the following, we consider simplified prefixes of up to 8 bits, with the leftmost bit in a prefix denoted by b_0, the next bit by b_1, etc. Only the case of ψ being a power of 2 is stated here.

Given 7 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 4 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, κ^{th} ROT-partition resides in LC_κ, where κ = 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. Non-local packets are delivered from their arrival LCs through the switching fabric to their respective home LCs for longest-prefix matching, if they

miss in the LR-caches of their arrival LCs. Each such a packet is routed across the switching fabric using b_2 and b_4 of its destination address. Note that P_3 belongs to every partition, because both b_2 and b_4 of P_3 are ì*î (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 ì*î. On the other hand, the use of b_0 and b_4 for partitioning arrives at $\{P_3, P_7\}$, $\{P_3, P_4\}$, $\{P_1, P_2, P_5\}$, $\{P_1, P_2, P_6\}$, where each partition involves 2 or 3 prefixes. Obviously, 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 ì0î (or ì1î) and Φ_* is the number of prefixes with b_v bits being ì*î (since these prefixes are 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 $0 \le v \le 31$, where Φ is the set size (a fixed number). 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% for the set of prefixes obtained from [2]) have length no more than 24, and b_v in a prefix is ì*î 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 $0 \le v \le 31$. When examining bit b_v, this criterion calls for ignoring prefixes with bit b_v = ì*î, counting only those with bit b_v being ì0î or ì1î. Notice that Criterion (1) considers only Φ_* whereas Criterion (2) deals with Φ_0 and Φ_1. In general, when multiple control bits are to be chosen for a set of prefixes, these two criteria are applied recursively, first to find one control bit b_v among $0 \le v \le 31$, and then to find 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.

3.2 LR-Cache Operation and Organization

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 ñ 10 *ms* [3, 15]. 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 this simple flushing will not work effectively if the routing table is updated incrementally and very frequently.) 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 ìmixî) status bit, to indicate its LOC/REM status. This bit is necessary for implementing an efficient cache replacement mechanism, realizing a suitable ìmixî 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, 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%. 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 4 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, Ö, 1111_2, in parallel via the ìXORî 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 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.

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 [5]. The degree of set

associativity for LR-caches is chosen 4, and this choice leads to nearly best performance, according to our simulation results and an earlier work [16]. The cache size ranges from 1K to 8K blocks.

A *victim cache* is for keeping blocks which are evicted from a cache due to conflict misses. It 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 mechanism for replacement (e.g., LRU, FIFO, random). When a packet is searched in an LR-cache, its associated victim cache is also examined simultaneously. In this study, each LR-cache is equipped with a victim cache of 8 blocks, which are found to yield 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 ì XORî logics, shown in Fig. 2) 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). 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.

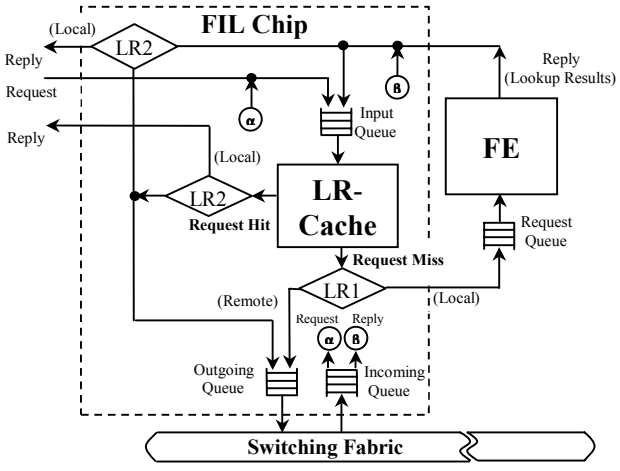

Fig. 2. LR-cache and relevant logics housed inside the FIL chip, where an LR1 (or LR2) detector examines the destination (or originator) address.

If a remote lookup is to be performed (by detector LR1), the packet is moved to the Outgoing Queue (see Fig. 2) 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. 2) 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 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 because of good parallelism in packet lookups.

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.

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 [12] (called RT_1) and the other with 140,838 prefixes [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. Following the two criteria for table partitioning stated in Sec. 3.1, we arrived at the two desired bit positions for fragmenting RT_1 (or RT_2) into

4 partitions: bits 12 and 14 (or 8 and 14). If RT_1 (or RT_2) is to be partitioned into 16 fragments, the preferred partitioning bit positions are found to be 12, 14, 15, 16 (or 11, 13, 14, 16).

Storage Sizes

Three distinct tries (namely, the DP trie [8], the Lulea trie [7], and the LC trie [12]) 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 1082, 1082, 1042, and 1027 Kbytes), respectively, assuming that each node in the trie consists of one byte for the index field plus 4 bytes for each of the five pointers. Since the trie size before partitioning is 859 (or 2827) Kbytes, the DP trie sees the amount of SRAM reduction in each LC caused by partitioning to exceed 638 (or 1745) 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 279 and 335 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 2492) 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. 3 under DP_S (or DP_W).

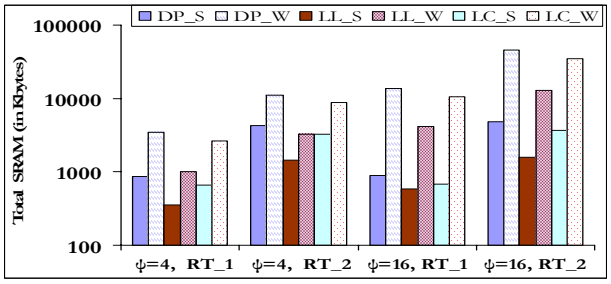

Fig. 3. Total SRAM (in Kbytes) required for different tries under IPv4.

For the Lulea trie [7] (whose storage requirement is often the lowest) with $\psi = 4$ under RT_1 (or RT_2), the partitioned tables in 4 LCs require 90, 91, 89, and 87 Kbytes (or 361, 358, 348, and 342 Kbytes), respectively, as opposed to roughly 260 (or 822) 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 461) 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 113) Kbytes in any LC after partitioning RT_1 (or RT_2), giving rise to a savings of at least 221 (or 709) 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. 3. Similarly, the LC-trie [12] under RT_1 (or RT_2) enjoys storage reduction in any LC due to SPAL by no less than 815 (or 1345) Kbytes, for $\psi = 4$ with a fill factor of 0.25. The storage saving amount in an LC increases to over 1025 (or 1927) 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.

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, Ciscoís 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 [14, 17], one stream for each LC. For Abilene-I, Abilene-II, Bell Labs-I, 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 [14]), 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 April 30th till July 26th, 1998 with more than 1.35 billions requests in total [17]), 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. Under the clock cycle of 5 *ns* simulated for the LC speed of 40 (or 10) Gbps, one packet was generated in anywhere from 2 cycles to 18 cycles (or from 6 cycles to 74 cycles). Given a trace, once a packet is generated at an LC, its destination was supplied by the trace. 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 [15] 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. 2); 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 [7] requires 6.2 (or 6.6) memory accesses per lookup on an average for RT_1 (or RT_2), while the DP trie [8] 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), we assume the memory access time as 12 *ns* and the code execution time as 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 Outcome 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 and 2002 available to the public [14, 17]. The results are for different cases: 10 Gbps & 40-cycle lookup, 10 Gbps & 62-cycle lookup, 40 Gbps & 40-cycle lookup, and 40 Gbps & 62-cycle lookup. Since those cases see their results follow a similar trend, in this article, we present the simulation outcomes only for *the case of 40 Gbps & 40-cycle lookup*, under two traces from WorldCup98, namely, D_75 (for July 9, 1998) and D_81 (for July 15, 1998), two traces from the Abilene-I data set in the PMA Long Traces Archive, namely, L_92-0 and L_92-1, and the Bell Labs-I trace from the same archive.

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 ìmixî value (γ, reflecting % of blocks devoted for REM results) affected lookup performance under various LR-cache sizes (β, in blocks) for ψ ranging from 1 to 16. The simulation outcomes all follow a trend similar to that signified in Fig. 4, where the mean lookup time (in cycles of 5 *ns* each) as a function of γ under the five traces is depicted for $\psi = 4$ with β equal to 4K. This figure reveals that $\gamma = 50\%$ typically yields best (or nearly best) performance. Note that if the LR-cache is small (like $\beta =$ 1K), $\gamma = 25\%$ may then be chosen, namely, only one cache block per set is for the REM results. Hence, we present subsequently the outcomes only for $\gamma = 50\%$ (or 25%) under $\beta \geq 2K$ (or $\beta = 1K$).

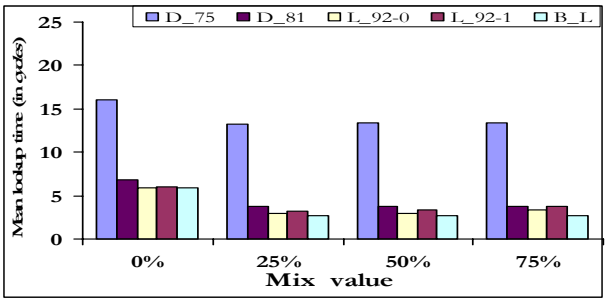

Fig. 4. Mean lookup time (in cycles) versus mix value (γ) for $\psi = 4$ and $\beta = 4K$.

We investigated the impact of the LR-cache size on SPAL performance, with simulation results under the five traces demonstrated in Fig. 5 ñ Fig. 6. The average lookup time (in cycles) as a function of the cache size (β) for $\psi = 16$ is depicted in Fig. 5, 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; with $\beta = 4K$, the mean lookup times under SPAL sized $\psi = 16$ drop below 9.2 cycles for all the traces shown, translating to a lookup speed beyond 21 million packets per second for each LC. Therefore, a SPAL-based router sized 16 can forward over 336 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 $\psi = 16$ accelerates packet lookups by 4.2 times, when compared with its commercial counterpart, despite its reduced total SRAM amount and a possibly shorter worst-case lookup time.

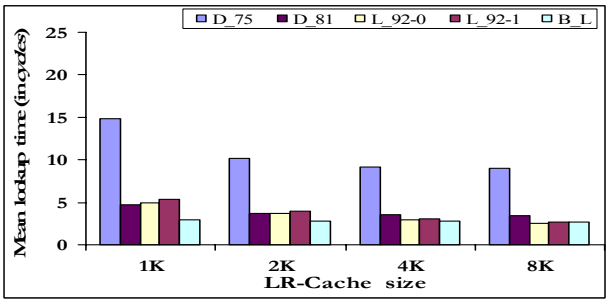

Fig. 5. Mean lookup time (in cycles) versus LR-cache size (β) under $\psi = 16$.

Mean lookup performance versus ψ (i.e., the number of LCs) under $\beta = 4K$ and $\gamma = 50\%$ is illustrated in Fig. 6. According to the figure, a larger ψ generally leads to a lower 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 L92_0 drops from more than 6 cycles down to less than 3 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 [6]), the mean lookup time will be independent of ψ and be always equal to that of ψ = 1 depicted in Fig. 6. 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.

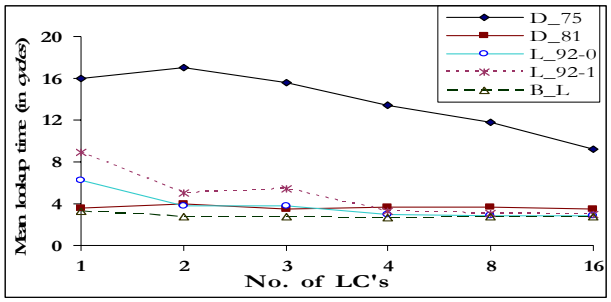

Fig. 6. Mean lookup time (in cycles) versus ψ under β = 4K and γ = 50%.

6. Conclusion

A speedy packet lookup (SPAL) technique has been investigated for high-performance routers, realized by fragmenting the BGP routing tables and incorporating a small cache (say 4K◊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 SPAL exhibits over 4.2 times faster mean lookups in a router with ψ = 16 LCs under the LR-cache size of 4K blocks, when compared with 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. The partitioning bits are chosen according to the prefixes in a given routing table and irrespective of packet streams, and the partitioning bits obtained this way often yield high lookup concurrency to arrive at a small mean lookup time. In addition, SPAL is feasibly applicable to IPv6. With its ability to speed up packet lookup performance while lowering overall SRAM substantially, SPAL is ideally suitable for future high-performance routers.

References

[1] M. Akhbarizadeh and M. Nourani, ìAn IP Packet Forwarding Technique Based on Partitioned Lookup Table,î *Proc. 2002 IEEE International Conf. on Communications*, Apr./May 2002.

[2] AS1221 BGP Table Data, URL ñ http://bgp.potaroo.net/as1221/bgp-active.html, routing table snapshot taken at 4:14pm, Jan 30, 2003.

[3] A. Basu and G. Narlikar, ìFast Incremental Updates for Pipelined Forwarding Engines,î *Proc. IEEE Conf. on Computer Communications (INFOCOM ì03),* Apr. 2003.

[4] BGP Table Data, URL ñ http://bgp.potaroo.net, 2003.

[5] T. Chiueh and P. Pradhan, ìHigh-Performance IP Routing Table Lookup using CPU Caching,î *Proc. IEEE Conf. on Computer Communications (INFOCOM ì99),* Apr. 1999, pp. 1421-1428.

[6] T. Chiueh and P. Pradhan, ìCache Memory Design for Internet Processors,î *IEEE Micro*, vol. 20, Jan./Feb. 2000.

[7] M. Degermark *et al.*, ìSmall Forwarding Tables for Fast Routing Lookups,î *Proc. ACM SIGCOMM 1997 Conference*, Sept. 1997, pp. 3-14.

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

[9] C. Estan and G. Varghese, ìNew Directions in Traffic Measurement and Accounting,î *Proc. ACM SIGCOMM 2002 Conference*, Aug. 2002, pp. 323-336.

[10] P. Gupta, S. Lin , and N. McKeown, ìRouting Lookups in Hardware at Memory Access Speeds,î *Proc. IEEE Conf. on Computer Communications (INFOCOM ì98)*, Apr. 1998, pp. 1240-1247.

[11] G. Huston, ìAnalyzing the Internetís BGP Routing Table,î *The Internet Protocol Journal*, vol. 4, no. 1, Mar. 2001.

[12] S. Nilsson and G. Karlsson, ìIP-Address Lookup Using LC-Tries,î *IEEE J. on Selected Areas in Communications*, vol. 17, no. 6, pp. 1083-1092, June 1999.

[13] Pericom Semiconductor Corporation, ìThroughput Expansion with FET-Based Crossbar Switching,î Nov. 2001, URL ñ http://www.pericom.com/.

[14] PMA Long Traces Archive, URL ñ http://pma.nlanr.net/Traces/long/, Passive Measurement and Analysis, National Laboratory for Applied Network Research, Sept. 2002.

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

[16] N. Tzeng, ìHardware-Assisted Design for Fast Packet Forwarding in Parallel Routers,î *Proc. 2003 International Conference on Parallel Processing*, Oct. 2003, pp. 11-18.

[17] WorldCup98 Dataset, URL ñ http://ita.ee.lbl.gov/html/contrib/WorldCup.html, The Internet Traffic Archive, Lawrence Berkeley National Laboratory, Apr. 2000.