Reducing Cache Miss Ratio For Routing Prefix Cache

Huan Liu
Department of Electrical Engineering
Stanford University, CA 94305
huanliu@stanford.edu

***Abstract*—Because of rapid increase in link capacity, an Internet router has to complete routing lookup function in a much shorter amount of time in order to keep up with the line rate. Many fast routing lookup algorithms have been proposed in the past. Because most of them require some memory accesses, their lookup speed is limited by memory access speed. IP address caching has been used to improve upon existing routing lookup algorithms by storing the most recently referenced IP address. Most recently, Routing Prefix caching has been proposed to increase caching effectiveness. In this paper, we evaluate techniques to further reduce cache miss ratio, thereby, increase caching effectiveness for a routing prefix cache. We first study the effect of routing table compaction, which reduces the routing table size. Then we propose and evaluate a new cache replacement policy that outperforms both least recently used and least frequently used replacement policy. The experiment result shows that up to 39% percent reduction in cache miss ratio could be achieved. It further proves that routing prefix cache has the potential to greatly outperform IP address cache.**

I. Introduction

The explosive growth of Internet leads to rapid deployment of faster and faster fiber links in the backbone. Currently, OC-192 links (10Gbit/s) are starting to be widely deployed, and very soon, OC-768 links (40Gbit/s) will be commercially available. Current research in Dense Wave Division Multiplexing (DWDM) technology shows even more impressive result, hundred of channels are packed into one single fiber, resulting in several terabits per second capacity. Meanwhile, in the electrical domain, the processing speed has not improved as impressively. Especially the memory system bandwidth and latency are rapidly falling behind.

In order to keep up with increasing link speed, the router has to employ novel architecture and algorithms to complete routing function in a much shorter period of time. In a base configuration, an IP router needs to inspect destination IP address in every packet header, then it will lookup routing table for next hop address in order to determine where to forward the packet.

It turns out that routing lookup increasingly becomes a bottleneck in system design. To complicate things further, Classless Inter-Domain Routing (CIDR) [1] was proposed and adopted in an attempt to slow the exhaustion of Internet Protocol (IP) address space. Now, instead of finding an exact match in the routing table, the router needs to perform Longest Prefix Matching (LPM) based on the destination IP address. The router needs to maintain a table of prefixes rather than exact IP addresses. Upon address lookup, the longest prefix that matches the beginning of the IP address is chosen.

Various algorithms have been proposed to efficiently perform longest prefix matching operation, including software based schemes such as [2], [3], [4], and hardware based schemes such as[5], [6], [7], [8], [9]. Software based scheme typically requires at least 4 to 6 memory access for each address lookup, so they are relatively slow. Hardware based schemes can speed up the lookup operation by reducing the number of memory accesses required. In particular, Ternary Content Addressable Memory (TCAM) requires only one single memory access. Still, it may not be fast enough because memory access speed is lagging the capacity growth.

IP address caching has been used in routers to speed up lookup operation over existing routing lookup algorithms. It stores the most recent lookup result in a local fast storage in hope that it will be shortly accessed again. Caching on destination IP address worked well in the past, however, rapid growth of Internet will soon reduce its effectiveness. Unless the cache size is uneconomically large, there may be little locality that could be exploited. Several improvements [10], [11] were proposed to improve caching effectiveness. Most recently, routing prefix caching has been proposed [12] to fully exploit temporal locality. Instead of caching a single IP address, the routing prefix is cached instead. Because of the aggregation of routing prefix, we need a much smaller cache to achieve the same cache miss ratio as an IP address cache.

Even though prefix cache outperforms IP address cache, the overall cache miss ratio may still be too high especially in the Internet backbone, where a large number of flows exist. The problem is exacerbated by the desire to keep the cache size small. A smaller cache not only reduces cost, but it also makes it possible to speed up the circuit by reducing capacitance loading on the cache lines.

In this paper, we consider the case where a small routing prefix cache is used. Instead of building uneconomically large cache, we evaluate and propose simple and cost effective techniques to improve caching effectiveness. First, we evaluate the impact of routing table compaction techniques [13]. A compacted routing table will allow several routing prefixes to use the same cache entry, therefore, further aggregation is achieved. Secondly, we propose and evaluate a new cache replacement policy called multi-segment Least Recently Used (mLRU) policy. We show that our proposed replacement policy outperforms both Least Recently Used (LRU) and Least Frequently Used

(LFU) replacement policy.

In this following, we first describe the routing prefix cache architecture as proposed in [12]. Then we describe the two techniques to reduce cache miss ratio. Lastly, we will present simulation results.

II. Routing Prefix Cache Architecture

A routing lookup application behaves very differently from programs running on a general purpose microprocessor. Therefore, different cache design choices are necessary. Specifically, our proposed routing prefix cache will have only one entry in a cache line, and will use a fully associative design. The rational is as follows.

First of all, there is little or no spatial locality to exploit in a packet stream. When IP address x is being looked up, the next IP address to be looked up has equal chance to be $x + 1$ or any other random IP address. Therefore, a large cache line size does not reduce the cache miss ratio. Indeed, it has been argued [10] that the cache line size should be set to one entry.

Secondly, a fully associative routing prefix cache greatly outperforms a set associative cache, again because of the lack of spatial locality. Cache in microprocessor typically uses a set associative design. Namely, a memory location can only map into a certain set instead of the whole cache. A set associative cache provides almost the same performance as a fully associative design because of the sequential nature of a program. Unfortunately, routing lookup cache does not enjoy this sequential nature at all. The performance difference is sufficiently large to warrant the more complex hardware involved.

We note that a cache with one entry cache line size and fully associative design is in essence a Content Addressable Memory (CAM), since the lookup key is compared against all tags at the same time. An IP address cache with the proposed design will be very similar to binary CAM as each tag stores a fixed length (32 bits for IPv4) IP. In contrast, a prefix cache with the proposed design is very similar to a Ternary CAM because it can specify variable length tag by changing the mask. The implementation of routing prefix cache should be straightforward given the similarity.

The prefix cache architecture is shown in Figure 1. The packet processing engine is responsible for all packet processing functions. It could be either a dedicated ASIC or a network processor. At the minimum, it will extract destination IP address from the packet header, then consult the routing table to determine its next hop address. The packet processing engine integrates a micro engine and a cache on the same chip. Route lookups that missed the on-chip cache will be sent off-chip to the external TCAM for full routing table lookup. The reason that we use TCAM for full routing table lookup instead of any other routing lookup algorithms is because one of the routing table compaction techniques we will use takes advantage of a special property of TCAM.

III. Routing Table Compaction

A simple way to improve caching effectiveness is to reduce the size of the routing table. A smaller table means there is higher chance that any single entry will be accessed shortly again. It represents another level of aggregation.

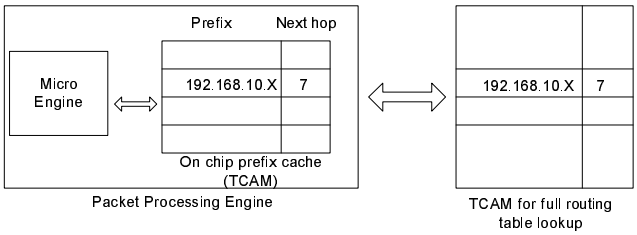

Fig. 1. Prefix cache architecture

A couple of techniques for reducing the routing table size are proposed in [13]. We briefly review the two techniques here. The first technique is to reduce redundant routing table entries. In the example shown in Figure 2, P_2 is a redundant entry. When P_2 is removed, any prefix that will match P_2 will match P_1 instead. Since both P_2 and P_1 correspond to the same port (next hop), any packet will still be routed correctly even if P_2 is removed.

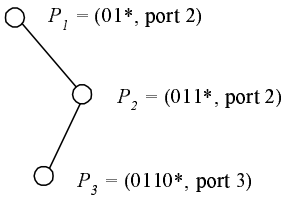

Fig. 2. A pruning example.

The second technique — mask extension — takes advantage of a special property of Ternary Content Addressable Memory (TCAM), so that it is possible to store several prefixes in the same TCAM entry. We say two prefixes are *compatible* if both correspond to the same routing nexthop. Only compatible prefixes can be combined into a single routing entry. For example, if there are two prefixes: 1000 and 1100, they could be combined into 1-00 where - is used to denote don't care bit. Since TCAM is capable of storing don't care in any bit position, 1-00 will use only one entry in the TCAM. The mask extension technique essentially becomes a logic minimization problem, which could be solved by efficient heuristic algorithms.

The two techniques are very effective at compacting a routing table and could reduce a routing table size by up to 48% as shown in [13].

IV. A New Cache Replacement Policy (MLRU)

Another way to improve caching effectiveness is to design optimized cache replacement policy. The most widely used cache replacement policies, such as Least Recently Used (LRU) and Least Frequently Used (LFU), are not optimized for routing prefix cache. They are specifically designed for cache in microprocessors where both spatial locality and temporal locality could be exploited. They work well because typical program execution is very localized. A small portion of the code (called active code) is executed most of the time. LFU policy tries to

identify active code by keeping track how often they are accessed. LRU policy tries to identify active code by assuming the most recently accessed code is active code.

The reason that LRU and LFU do not work well for routing prefix cache is because routing lookup application is sufficiently different from a program running on general purpose microprocessor. Specifically:

1) Packet stream that goes through routing prefix cache does not have the inherent sequential nature. The sequence of IP address appears to be random. So, there is only temporal locality to exploit, no spatial locality at all.
2) Packet stream is bursty, which mean there are lots of packets targeting the same destination in a short time period, then there maybe no packet targeting that destination for a while.

The problem with LRU is that it always stores an entry at the beginning of the cache, even if only one packet is sent to an IP address. It takes a long time for the entry to age out of the cache. In contrast, LFU initially stores the entry close to the end of the cache, making it hard to stay in cache before the next reference. Ideally, a new entry should not be placed at the beginning or end of cache, but rather some place in the middle. If it is for a short burst, the entry can quickly age out of the cache. But if it is for a long burst, the entry can gradually move towards the beginning as it is referenced, therefore it can not be easily aged out in the future.

We propose a new cache replacement policy that combines the advantage of both LRU and LFU. We call it multi-segment Least Recently Used (mLRU) policy. The idea is to break a cache into N separate cache segments. When an entry e is accessed, either as a result of cache miss or cache hit, its access frequency $f(e)$ determines which segment it is inserted into. Within the segment, the entry is always inserted at the beginning. An entry ages naturally when new entries are inserted ahead of it. When there are sufficiently large number of insertions ahead of an entry before it is accessed again, the entry will be aged out of the cache.

An example mLRU cache with 4 segments is shown in Figure 3. Because the entry can only be inserted at the beginning of a cache segment, there are only 4 valid insertion points. Segment 4 contains only entries with their access frequency greater than 4. Segment 3 contains only entries with their access frequency greater than 3, and so on.

When a routing prefix P is first referenced, a cache miss will result. When P is loaded into the cache, it will be stored at the beginning of segment 1 and its frequency is set to 1. All other entries in Segment 1 shifts down, and the last entry will be shifted out of the cache. As other prefixes are inserted into the cache when they are being accessed, P will quickly age out of the cache. If, however, P is referenced again before it is aged out of the cache, it will be inserted at the beginning of segment 2.

In general, whenever a prefix is referenced while it is still in the cache (cache hit), its frequency count is incremented. Based on the new frequency count, the segment is determined, and the prefix is inserted at its beginning. All entries between the old position and the new position are shifted down.

For long burst, the mLRU policy behaves like LRU policy.

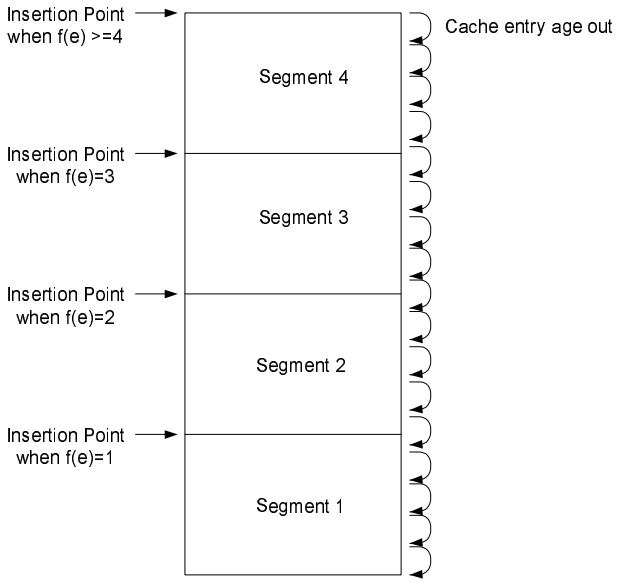

Fig. 3. An mLRU cache with 4 segments. An entry can only be inserted at the beginning of a segment based on its frequency. All entries after the newly inserted entry are shifted down.

Each referenced prefix is placed at the beginning of the cache, and the flow has to be idle for a long while before it is aged out of the cache.

For short burst, the mLRU policy behaves like LRU policy with a much smaller cache size. This is to discourage short flow. The rational is intuitive; caching long flow is the most beneficial in terms of overall caching effectiveness.

We argue that cache with mLRU policy is no harder to implement than cache with LFU policy. Both cache have to keep track of the reference frequency of each entry. But in terms of entry insertion, mLRU is simpler to design because there are only a limited number of insertion points, and also no sorting is needed at all.

V. Simulation Result

To evaluate the effectiveness of our proposed techniques to reduce cache miss ratio, we simulated three separate traces collected by NLANR MOAT team [14] from different Internet access points over OC3 links, and compared the result with those using prefix caching alone [12].

Because the traces do not have associated routing table snap shot, we used MaeEast routing table captured by IPMA [15] for the evaluation. Some of the destination IP addresses do not correspond to any valid routing prefix as a result of the mismatch, so we omit them from our simulation. The number of valid IP addresses in the traces range from 1.8 million to 12.8 million. The details of these traces are shown in Table I.

TABLE I

Characteristics of the three packet traces

Trace Name	# of valid IP	# of unique IP
970222	633156	10197
SDC958	1857296	13846
SDC964	12838026	54022

A. Effect of routing table compaction

We first evaluate the effectiveness of routing table compaction techniques. We first apply pruning technique to remove redundant prefixes in the routing table. Then we apply complete prefix tree expansion in order to guarantee correct routing lookup result [12]. As a result of complete prefix tree expansion, a routing lookup will only match a leaf node in the routing prefix tree. Any internal prefix node will not have any effect. Therefore we remove all internal prefix nodes to save space. Lastly, since we use TCAM to store the full routing table, we apply the mask extension technique to combine several prefixes into a single TCAM entry. The routing table snapshot we choose has 25160 routing prefixes. After all transformation, we end up using only 18598 entries in the TCAM, a 26% space saving. Note that this number is smaller than that reported in [12] because we have to apply complete prefix tree expansion in order to guarantee correct lookup result.

We compare cache miss ratio based on compacted routing table with those based on original routing table. In order to enable a fair comparison, we use LRU replacement policy for both cache simulation. For brevity, result for only the sdc958 trace is shown in Figure 4.

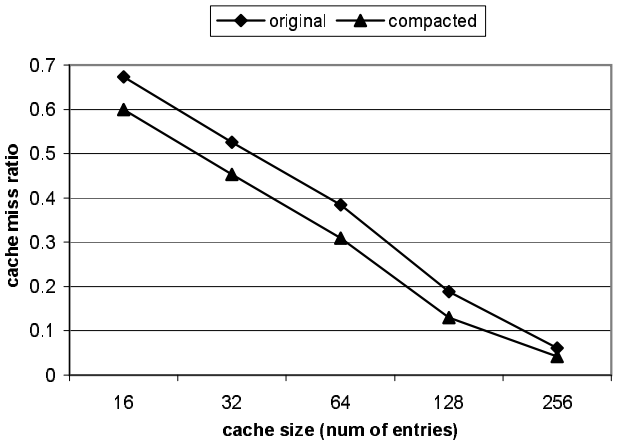

Fig. 4. Cache miss ratio when using original routing table and compacted routing table for trace sdc958.

As expected, caching based on compacted routing table consistently outperform caching based on original routing table. The difference between the two shrinks as the cache size grows larger. This is because as cache size grows, the cache miss ratio gets closer to the inherent limit, therefore, improving upon it gets harder

B. Effect of mLRU cache replacement policy

To evaluate mLRU cache replacement policy's effectiveness, we first compact the routing table. Then we apply LRU, LFU and mLRU replacement policies. The result for trace sdc958 is shown in Figure 5. For mLRU, we arbitrarily choose to partition the cache into 16 equal size segments. Therefore, we only need 4 bits to record frequency for each entry. It is a very minimal overhead.

As shown, LRU outperforms LFU by a wide margin. This is understandable because LRU more directly exploit temporal

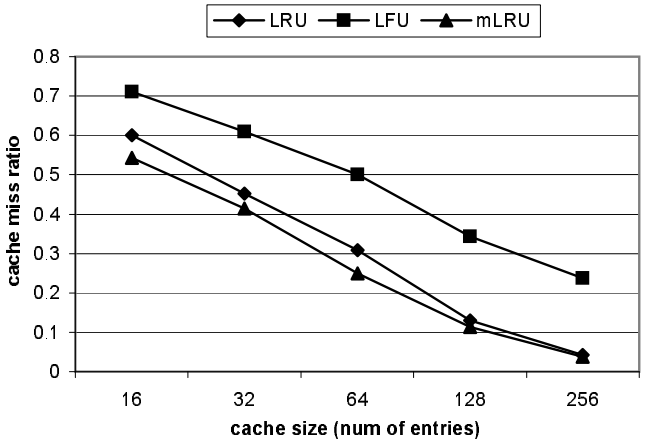

Fig. 5. Cache miss ratio comparison between LRU, LFU and mLRU policy for trace sdc958. Compacted routing table is used. 16 cache segments assumed for mLRU.

locality. Since only temporal locality is available in a packet stream, a replacement policy directly exploiting it should perform better.

It is also clear from Figure 5 that our proposed mLRU algorithm outperforms both LRU and LFU. Since mLRU combines benefits from both LRU and LFU policy, it is more optimized for routing lookup cache. The gap between LRU and mLRU shrinks again because of the inherent limit on cache miss ratio.

C. Total reduction in cache miss ratio

We applied both the routing table compaction techniques and the new replacement policy on routing prefix cache to reduce the cache miss ratio. Again, we assume mLRU partitions the cache into 16 equal segments. The simulation result for all three traces is shown in Table II, III, IV.

TABLE II
CACHE MISS RATIO FOR TRACE SDC964

# of cache entries	16	32	64	128	256
original	59.6%	47.3%	32.6%	14%	5.8%
compaction+mLRU	50.2%	35.3%	21.1%	9%	3.7%
% reduction	15.7%	25.5%	35.5%	35.2%	35.9%

TABLE III
CACHE MISS RATIO FOR TRACE SDC958

# of cache entries	16	32	64	128	256
original	67.4%	52.5%	38.4%	18.8%	6.2%
compaction+mLRU	54.3%	41.4%	25%	11.4%	3.7%
% reduction	19.4%	21.1%	35%	39.4%	39.6%

TABLE IV
CACHE MISS RATIO FOR TRACE 970222

# of cache entries	16	32	64	128	256
original	48.1%	37%	25.6%	13.9%	7.2%
compacton+mLRU	42.7%	30.5%	19.6%	10.9%	5.6%
% reduction	11.2%	17.5%	23.6%	21.9%	22.8%

As shown in the tables, our proposed techniques can significantly reduce the cache miss ratio. The total reduction range from 11% up to 39.6%. As a result, significant routing lookup speed improvement can be achieved with little extra cost.

VI. Conclusion

In this paper, we evaluated techniques to reduce cache miss ratio on routing prefix cache. First, we propose to compact routing table. A smaller table means any prefix will have a higher chance to stay in the cache when a new prefix is inserted. Then, we propose a new cache replacement policy, called multi-segment Least Recently Used (mLRU) policy, that combines benefits of both LRU and LFU. We show by simulation that our proposed techniques can significantly reduce cache miss ratio with little extra cost.

The result shows that routing prefix cache has the potential to achieve much lower cache miss ratio. It confirms that it is a better choice compared with IP address cache commonly used to speed up routing lookup.

VII. Acknowledgment

The author would like to thank the MOAT PMA group at the National Laboratory for Applied Network Research (NLANR) for providing free access to the trace data under National Science Foundation NLANR/MOAT Cooperative Agreement (No. ANI-9807479).

References

[1] Y. Rekhter and T. Li, "An architecture for ip address allocation with cidr," *RFC 1518*, 1993.

[2] S. Nilsson and G. Karlsson, "Ip-address lookup using lc-tries," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1083–92, 1999.

[3] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high-speed ip routing lookups," in *Proc. ACM SIGCOMM*, Cannes, France, 1997, pp. 25–36.

[4] A. Brodnik, S. Carlsson, M. Degermark, and S. Pink, "Small forwarding tables for fast routing lookups," in *Proc. ACM SIGCOMM*, Cannes, France, 1997, pp. 3–14.

[5] P. Gupta, S. Lin, and N. McKeown, "Routing lookups in hardware at memory access speeds," in *Proc. Infocom*, San Francisco, Apr. 1998.

[6] M. Kobayashi, T. Murase, and A. Kuriyama, "A longest prefix match search engine for multi-gigabit ip processing," in *Proc. ICC*, June 2000.

[7] T.B. Pei and C. Zukowski, "Vlsi implementation of routing tables: Tries and cams," in *Proc. Infocom*, 1991.

[8] A. McAuley and P. Francis, "Fast routing table lookup using cams," in *Proc. Infocom*, Mar. 1993, vol. 3, pp. 1382–91.

[9] T. Hayashi and T. Miyazaki, "High-speed table lookup engine for ipv6 longest prefix match," in *Proc. Globecom*, 1999, vol. 2, pp. 1576–1581.

[10] T.C. Chiueh and P. Pradhan, "Cache memory design for network processors," in *Proc. High Performance Computer Architecture*, 1999, pp. 409–418.

[11] B. Talbot, T. Sherwood, and B. Lin, "Ip caching for terabit speed routers," in *Proc. Globecom*, 1999.

[12] H. Liu, "Routing prefix caching in network processor design," in *Proc. ICCCN*, Phoenix, AZ, 2001.

[13] H. Liu, "Reducing routing table size using ternary-cam," in *Proc. Hot Interconnect 9*, Stanford, 2001.

[14] National Laboratory for Applied Network Research, *Passive Measurement and Analysis project*, http://moat.nlanr.net/pma.

[15] Merit Networks, Inc., *IPMA project*, http://www.merit.edu/ipma.