

Routing Prefix Caching in Network Processor Design

Huan Liu

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

Abstract— Cache has been time proven to be a very effective technique to improve memory access speed. It is based on the assumption that enough locality exists in memory access pattern, i.e., there is a high probability that an entry will be accessed again shortly after. However, it is questionable whether Internet traffic has enough locality, especially in the high speed backbone, to justify the use of cache for routing table lookup. We believe there is enough locality if routing prefixes are cached instead of individual IP addresses. This paper is the first to evaluate effectiveness of caching on routing prefix. We propose an on-chip routing prefix cache design for the network processor. The control-path processor is used in our design to fulfill missed cache requests. In order to guarantee correct lookup result, it is important that the control-path processor only returns cacheable routing prefixes. We present three implementations that all guarantee correct lookup result. Simulation result shows that our cache design performs much better than IP address cache, even after factoring in the extra complexity involved.

Index Terms— Cache, Longest Prefix Matching, CIDR, Ternary CAM, IP Lookup.

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 every packet header for destination IP address, 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][10]. Hardware based scheme which utilizes Ternary Content Addressable Memory (TCAM) has increasingly been used commercially to speed up the lookup operation. The key advantage is that it could return lookup result in one single memory cycle as compared to 4 or more memory accesses required by software based approaches. Clearly, to scale up to 10Gbps processing and beyond, TCAM is very advantageous because memory subsystem speed is lagging substantially.

The number of available routing prefixes is growing at a steady pace. A recent check of the routing table from University of Oregon Route Views Project [11] showed 61832 unique prefix entries. However, a typical router, even those at Internet eXchange Points (IXP), only sees a subset of those available prefixes. Still, the number of entries is large. When these prefixes are stored in a memory device, such as TCAM, the retrieval of routing information will be extremely slow because of the large capacitance on the wire. A smaller cache could be made faster simply because the capacitance load is much lower. It also permits usage of a more expensive but faster technology such as SRAM, which is infeasible for large amount of memory because of their lower density (SRAM is typically implemented using 6 transistors as compared to 1 transistor for DRAM). In addition, a smaller cache could be easily integrated on chip to the network processor, thus, saving the extra latency required for off-chip communication.

Most IP routers use caching to improve lookup speed by storing most recently used destination IP address in a fast local cache. The technique relies on there being enough locality in the traffic so that cache hit rate is sufficiently high, which in turn enables substantially lower average lookup speed. Caching technique based 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.

Some recent studies [12][13] on packet traces captured from enterprise class routers showed that there are sufficient

temporal locality to justify the use of IP address cache. Although, it is argued [14] that there will be little or no locality at all in a backbone router. It would seem intuitive that the faster the router operates at, the less likely there will be two packets with identical destination IP address in any given time interval.

Thus, it would seem useless to cache IP address in a high speed backbone router. But, it will still be very beneficial to cache routing prefixes. There are several reasons we believe this is true:

1. The possible number of IP routing prefixes is much smaller than the number of possible individual IP addresses. A recent check of routing prefixes from The University of Oregon Route View project[11] showed 61832 entries as compared to up to 2^{32} possible distinct IP addresses. The smaller range means there are higher probability that any one routing prefix will be accessed in short succession.

2. The number of routing prefixes seen at a router is even smaller. The reason is because CIDR is designed with geographical reach in mind. A border router for an Autonomy System (AS) is supposed to aggregate all routing prefixes within its domain. Thus a router would only see a small number of routing prefixes from each AS. Meanwhile, IP address does not benefit from the aggregation. It actually increases quickly as more computers are connected to the Internet.

3. Web content servers are increasingly being hosted in a central data center. They offer a cheaper solution by amortizing the cost of network maintenance and administration over a large number of clients. With World Wide Web dominating Internet traffic, it is not surprising that a large portion of the traffic will be moving towards data centers. These centers host thousand of web sites, but only share one single routing prefix. One would argue that routing prefixes for data centers should be permanently stored in the cache of any Internet routers.

4. Routing prefixes were assigned to Internet Service Providers (ISP) in blocks. For routing efficiency, ISP further allocates prefixes to different geographical locations. Thus a routing prefix typically represents a local area. Since traffic from New York will certainly peak at a different hour than those from Los Angeles, routing prefixes for those two areas will not be frequently used at the same time.

5. Caching routing prefix can benefit from routing table compaction techniques [15]. A compacted table is much smaller, resulting in higher probability that any cached routing prefix would be accessed again.

Given the above observations, it will be natural that routing prefix cache could produce better result than IP address cache. Furthermore, we believe routing prefix cache can more effectively deal with explosive Internet growth. As the number of Internet hosts grows, the number of connections increases. If IP address cache size remains the same, more addresses will be swapped in and out of the

cache resulting in higher miss ratio. On the contrary, routing table size grows very slowly. For example, MaeEast routing table size [16] has stayed roughly the same for the last five years. So, caching performance should stay roughly the same as well.

II. PREVIOUS WORKS

Most software based routing table lookup algorithms try to optimize usage of cache in general purpose processors, such as algorithms proposed in [17][18]. They carefully design the data structure to fit within a cache line, or fill several entries into the same cache line to facilitate several lookups per memory access. These algorithms concentrate on efficient use of cache in general purpose processor instead of designing alternative cache architecture for routing lookup application.

Talbot et al. [13] studied several traces captured from different access routers. A profiling of the address bit frequency is preformed to determine which bits should be used as cache index. Those indexing bits with frequency close to $\frac{1}{2}$ are chosen because they have equal probability to be either 1 or 0, thus cache reference is more evenly distributed. Caching based on carefully selected indexing bits showed very good locality in the IP traces captured. The result highly depends on the choice of indexing bits, which in turn, highly depends on the trace. The corresponding hardware implementation will prove to be ineffective if traffic condition changes.

Chiueh et al. [12] proposed two schemes to improve performance on IP address caching. The first scheme tries to shift address bits before indexing into the cache. Effectively, it uses fixed length prefix of IP address as the key, thus some level of address aggregation is achieved. The second scheme utilizes the fact that many routing prefixes share the same destination routing information, thus further aggregation of IP addresses is possible by carefully choosing the indexing bits, which may not necessary be contiguous. As a result, a smaller prefix could be generated, which leads to higher aggregation. Both of the proposed approaches need routing table dependent computation which could decrease the effectiveness as routing table keeps on changing.

Cheung et al. [19] formulated the problem of assigning part of routing table to different cache hierarchy as a dynamic programming problem, then showed placement algorithm so that the overall lookup speed is minimized. Besson et al. [20] also evaluated hierarchical cache in routers and its effect on routing lookup.

Our approach is different from all previous work in that we are trying to take advantage of the natural hierarchy of routing prefixes. Instead of caching IP address or any arbitrary portion of it, we cache the routing prefix as designated by the network operators. Our caching approach does not depend on routing prefix distribution, thus, it is applicable in a wider array of applications.

III. REFERENCE ROUTER ARCHITECTURE

We first describe the router architecture under which the cache system is employed. We consider a high speed router design where the data-path and control-path function are separated. The data-path is responsible for majority of the packet processing. At a minimum, it extracts different header fields, consults lookup table to determine its next hop, and then forwards packets to the appropriate interface through the switch fabric. Network Processor is used in the data-path to accomplish all processing required on a per packet basis. To improve address lookup speed, the network processor uses cache to store the most recent lookup results. If an address lookup matches entries in the cache (cache hit), the next hop information could be quickly returned. Because it is used to cache routing prefix, we refer to it as *prefix cache*.

The control-path typically uses a general purpose processor because the processing involved is much more complex. It performs exception handling, routing table update function etc. In addition, it maintains the complete routing table. When an IP lookup fails to find a match in the cache, it is forwarded to control-path processor to lookup the full routing table. The matching routing entry will be returned to network processor for caching. The control-path processor could use Trie data structure to store routing table in memory for efficient lookup, or alternatively, it could use TCAM to allow faster access. Because control-path serves as “main memory” for the prefix cache, we also refer it as *prefix memory* in this paper.

The reference router architecture is shown in Figure 1.

IV. PREFIX CACHE

Prefix cache is very different from cache used in general purpose processors. Its design is similar to TCAM. The tag in each cache entry contains two fields: routing prefix and its corresponding mask. The content of each cache entry is the next hop information, i.e., which interface of the router the packet should be forwarded to. An example prefix cache with two entries is shown in Figure 2. The cache is fully associative, i.e., the key is compared with all tags to determine a match. As such, no index is needed to search the cache as in set associative cache. We choose fully associative design because Internet traffic tends to

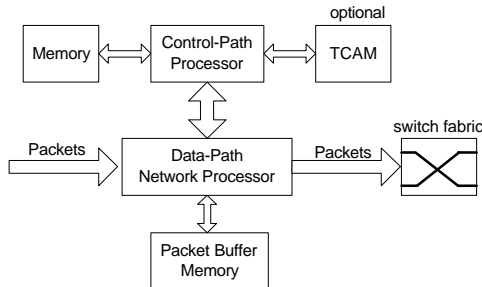


Figure 1: Reference Router Architecture

concentrate on certain portion of the routing table. Prefix cache will benefit little from the simplicity of a set associative design.

To describe how prefix cache works and what features it supports, we show by example each step involved in address lookup. The step number is marked on Figure 2 for clarity. Although not shown, the network processor is on the left making address lookup request, the prefix memory is on the right, serving lookup requests that missed the cache.

1. Prefix cache receives IP address when the network processor requests a lookup. Instead of exact matches, the IP address is first masked, and then compared with the routing prefix.

2. If the comparison succeeds (cache hit), the corresponding content containing next hop information is returned. No further search is needed. For example, IP address 128.10.15.3 will match the first entry, and port 8 is returned as lookup result.

3. If cache misses, i.e. if no matching routing prefix is found in the cache, the key (IP address) will be forwarded onto the prefix memory. For example, if lookup of IP address 128.11.1.6 is requested, it will fail to find a match in the prefix cache; subsequently, the request will be forwarded on.

4. Prefix memory looks up the key in its full routing table. Instead of returning only next hop information, the whole entry including routing prefix, mask and next hop is returned so that it can be cached for future references. It is inserted into the first available free entry in the cache. If no free entry is available, a replacement policy is used to determine which entry to free up. Note that, contrary to regular cache, no write back is required because network processor will never modify the prefix cache.

5. Prefix cache also supports invalidation command. When a routing entry is no longer valid, prefix cache needs to be notified. When invalidation command is issued by the prefix memory, only the prefix needs to be provided. As will be detailed in the next section, the prefix is guaranteed to match only one cache entry, thus the mask field is not needed to uniquely identify the entry to invalidate. In order to avoid unnecessary invalidation, prefix memory will keep track which entry is cached, and only invalidate when necessary. Note that this is the snoop logic in regular cache, but, since network processor will not modify prefix cache,

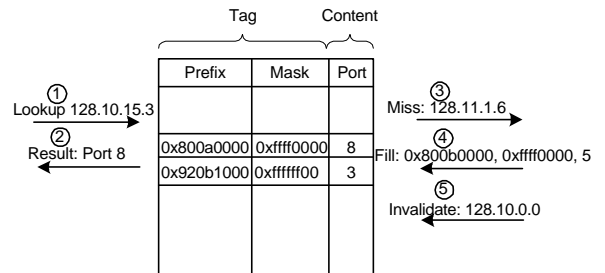


Figure 2: Example cache lookup flow

we could use the much simpler invalidation command instead.

Most of the functionalities of prefix cache can be found in regular cache or TCAM design. The implementation is straightforward, therefore, we do not elaborate further in this paper. Assume we use a 16 transistors per TCAM cell implementation, a 4K entry prefix cache will use $4k \cdot 16$ transistors/cell $\cdot 32\text{bits} = 2\text{M}$ transistors for its tag. It is small enough to be economically implemented on chip in today's technology.

V. PREFIX MEMORY

Prefix memory is implemented using control-path processor to reduce system cost. Assuming a 1% cache miss rate, and 5 million packet lookups per second (roughly corresponds to 10Gbit/sec line rate assuming average 2000 bits per packet as measured in [21]), only 50K lookups per second is required by the control-path processor. It is certainly well within today's general purpose processor's capability.

The main function of prefix memory is to serve address lookup that missed prefix cache. The prefix memory consults its own routing table, and returns a matching prefix for the prefix cache. The choice of what routing prefix to return is critical, as a regular routing prefix table could not be directly cached because an IP address could match multiple prefixes. If only one of those matches is cached, wrong lookup result could result. We present three designs, which differ mostly in how they transform the routing table so that correct lookup result is always guaranteed.

A. Complete Prefix Tree Expansion (CPTE)

We first define some terms. If an IP address matches two routing prefixes, we call the shorter one *subprefix*, and we call the longer prefix its *superprefix*. Intuitively, subprefix matches the beginning of its superprefix. Subprefix could not be cached, because if an IP address matches the subprefix, it could also match its superprefix, and if its superprefix is not cached, wrong lookup result could be returned. For example, in the left part of Figure 3, node A is a subprefix of node B. If node A is cached but not node B, an address lookup for 0101 would incorrectly return node A.

To guarantee correct lookup result, we transform the routing table into a *complete prefix tree* as first described in [4]. New nodes (routing prefixes) are created so that each node has either two child nodes, or none at all. The newly created node carries the same next hop information as its nearest ancestor. In the Figure 3 example, the transformation will create three more new nodes, each carrying the same next hop information as node A. Such a transformation guarantees that no IP lookup will match a subprefix, instead, a leaf node will be returned for caching.

Because of the expansion, CPTE could greatly increase routing table size, requiring a bigger memory or bigger

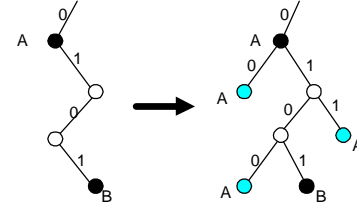


Figure 3: Complete prefix tree expansion. Three new nodes are created, each carries the same next hop information as node A.

TCAM to store it. We present two more alternatives that result in less routing table inflation.

B. No Prefix Expansion (NPE)

Certainly, if we do not perform complete prefix tree expansion, routing table size will remain the same. However, subprefix should not be cached to prevent lookup error. To accomplish this, any subprefix in the routing table should be marked as non-cacheable. For example, node A in the left part of Figure 3 should be marked as non-cacheable, because it is a subprefix of node B.

When cache misses in the network processor, prefix memory will perform matching against its full routing table. If it matches a non-cacheable entry, it should not return the prefix to fill the cache line. Instead, it should return the IP address (the key) with a complete mask to fill the cache entry. For example, in the left part of Figure 3, if a lookup for 0010 is performed, the matching routing entry will be node A. But since node A is not cacheable, instead of sending node A's prefix 0^* to be cached, the IP address itself (0010) along with a complete mask (1111) should be cached instead.

C. Partial Prefix Tree Expansion (PPTE)

In NPE, individual IP address will be cached when a non-cacheable prefix is matched. The prefix cache reduces to an IP address cache. The caching effectiveness will decrease as more and more IP addresses are cached.

As a trade off between CPTE and NPE, partial prefix tree expansion could be performed. In CPTE, most inflation is caused by prefixes being expanded into several levels. Instead of expanding each non-cacheable node to the lowest level, we choose to stop the expansion after the first level, i.e., it is only expanded once. An example is shown in Figure 4. Intuitively, the first level expanded node covers the largest address space, therefore, its effectiveness should be better

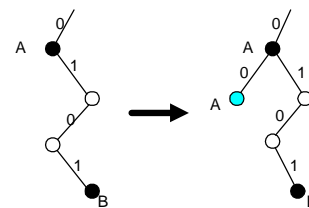


Figure 4: Partial Prefix Tree Expansion. Each non-cacheable node is only expanded one level.

than other expanded nodes. With PPTE, routing table inflation is reduced, and meanwhile, caching effectiveness is much better compared with IP address cache.

Similar to NPE, some IP address could still match a subprefix. In order to guarantee lookup correctness, these subprefixes should be mark as non-cacheable. When matched, the full IP address (the key) along with a complete mask should be cached instead of the subprefix.

D. Table Inflation Factor Comparison

We applied the three routing table expansions described to routing tables captured by IPMA[16] project. In addition we also compared with a more complete routing table captured by the University of Oregon Route View project (oix table). The result is shown in Table I.

As shown in the table, CPTE greatly increases the table size. In most cases, it grows about 50% in size; but for large table, it could grow more than 118%. In contrast, PPTE results in very modest expansion, typically limited to 20% growth. Thus it represents a very good compromise.

VI. SIMULATION RESULT

To evaluate the effectiveness of our routing prefix cache design, we obtained three separate traces collected by NLANR MOAT team[22] from several Internet access points over OC3 links. Although it is much more desirable to evaluate our design on traces captured in the backbone core, we could not find any trace publicly available.

Because the traces do not have associated routing table snap shot, we used MaeEast routing table captured by IPMA 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 II.

We implemented a cache simulator with least recently used (LRU) replacement policy, and evaluated the

performance of prefix cache along with the three prefix memory designs, namely, CPTE, PPTE and NPE. Unfortunately, we could not compare our design with other published result[12][13] on IP address caching because the traces used in those studies are not publicly available. Instead, we implemented simulator for IP address caching, and used that result as a comparison. The results are shown in Table III, Table IV and Table V. Cache miss ratio for IP address caching is shown in the row named "IP only". The top of each column shows the number of total cache entries in each simulation scenario. The result for trace SDC964 is also plotted in Figure 5.

We estimate prefix cache takes twice as much transistors to implement as an IP cache, because in addition to prefix storage and comparison logic, prefix cache also needs mask storage and masking logic. As shown in the tables and figure, prefix cache consistently outperforms IP cache, even after factoring in the extra complexity involved. Namely, prefix cache with half as many cache entries as IP cache has a smaller cache miss ratio in almost all cases. As expected, within the three prefix memory designs, CPTE outperforms PPTE, and PPTE in turn outperforms NPE.

The prefix cache is fully associative. For comparison, we also simulated set-associative prefix cache, where the first 8 bits of routing prefix are used as index to find the correct cache set. Prefix cache consistently outperforms IP address cache by a wide margin. However, the overall cache miss ratio is very high, ranging from 3% to 25%. The result is not shown in this paper because it is impractical. Set associative cache works very well for general purpose processors, the nature of sequential program helps to spread accesses evenly

TABLE I

ORIGINAL ROUTING TABLE SIZE, AND THE SIZE AFTER CPTE AND PPTE EXPANSION. THE EXPANSION PERCENTAGE IS ALSO SHOWN.

	Mae East	Mae West	Paix	Aads	PacBell	Oix
NPE	23554	32139	15906	29195	38791	61832
CPTE	34139	48875	58026	41846	23215	134930
	145%	152%	146%	143%	150%	218%
PPTE	26613	36662	18000	32541	44173	74363
	113%	114%	113%	111%	114%	120%

TABLE II

CHARACTERISTICS OF THE THREE PACKET TRACES

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

TABLE III

CACHE MISS RATIO FOR TRACE SDC964

# of entry	512	1024	2048	4096	8192
CPTE	1.9%	0.9%	0.38%	0.18%	0.1%
PPTE	2.3%	1.2%	0.5%	0.29%	0.17%
NPE	2.8%	1.6%	0.8%	0.48%	0.29%
IP only	4.9%	3%	2%	1.2%	0.8%

TABLE IV

CACHE MISS RATIO FOR TRACE SDC958

# of entry	512	1024	2048	4096	8192
CPTE	1.8%	0.77%	0.36%	0.27%	0.26%
PPTE	2.3%	1%	0.5%	0.34%	0.32%
NPE	2.8%	1.4%	0.7%	0.5%	0.4%
IP only	5.5%	2.5%	1.6%	0.95%	0.75%

TABLE V

CACHE MISS RATIO FOR TRACE 970222

# of entry	512	1024	2048	4096	8192
CPTE	4%	2%	0.88%	0.57%	0.57%
PPTE	4.4%	2.3%	1.1%	0.66%	0.66%
NPE	4.9%	2.7%	1.5%	0.8%	0.8%
IP only	9%	5.8%	3.7%	2.4%	1.6%

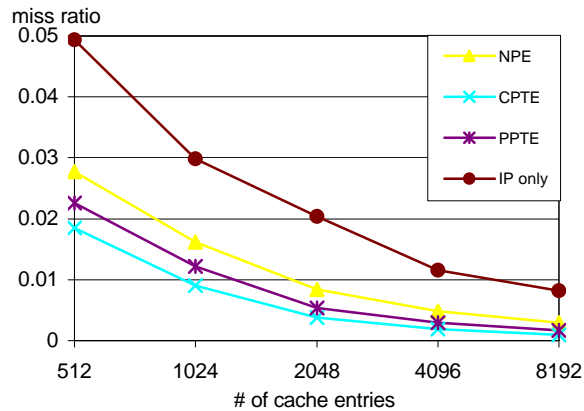


Figure 5: SDC964 cache miss ratio

across different cache set. However, Internet packet traces exhibit no such sequential nature at all (in terms of destination IP address), in fact, it tends to concentrate address lookup within a very small region of the routing table. Thus we believe set associative cache should not be used for IP address lookup.

VII. CONCLUSION AND FUTURE WORK

In this paper, we presented a routing prefix cache design for network processor. Our proposal will cache variable length routing prefix instead of fixed 32 bits IP address. Simulation result shows that our design works better than caching full IP address, even after factoring in the extra complexity involved. We believe our design will scale nicely with explosive growth of the Internet. The number of hosts in the Internet is growing rapidly, however, the number of routing prefixes is growing at a much slower and steady pace. We believe our scheme is the only way to achieve caching effectiveness without using uneconomically large cache for tomorrow's Internet.

Further study of our proposed design is still under way, especially its effectiveness in Internet backbone routers. We believe the advantage of our design will be more pronounced in the backbone. Simulation on long traces captured on a backbone router could validate our belief. It is also interesting to study the effect of routing table compaction technique. A smaller routing table may further improve cache effectiveness.

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, T. Li, "An Architecture for IP Address Allocation with CIDR," RFC 1518, 1993
- [2] S. Nilsson, G. Karlsson, "IP-address lookup using LC-tries," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1083-1092, 1999.
- [3] M. Waldvogel, G. Varghese, J. Turner, B. Plattner, "Scalable High-Speed IP Routing Lookups," *Proc. ACM SIGCOMM 1997*, pp. 25-36, Cannes, France.
- [4] A. Brodnik, S. Carlsson, M. Degermark, S. Pink, "Small Forwarding Tables for Fast Routing Lookups," *Proc. ACM SIGCOMM 1997*, pp. 3-14, Cannes, France.
- [5] P. Gupta, S. Lin and N. McKeown, "Routing Lookups in Hardware at Memory Access Speeds," *Proc. Infocom*, April 98, San Francisco.
- [6] M. Kobayashi, T. Murase and A. Kuriyama, "A Longest Prefix Match Search Engine for Multi-Gigabit IP Processing", *Proc. ICC 2000*, June 2000.
- [7] D. Shah and P. Gupta, "Fast Updates on Ternary-CAMs for Packet Lookups and Classification," *Proc. Hot Interconnects VIII*, August 2000, Stanford.
- [8] T.B. Pei and C. Zukowski, "VLSI Implementation of Routing Tables: Tries and CAMs," *Proc. Infocom 91*
- [9] A. McAuley and P. Francis, "Fast Routing Table Lookup Using CAMs", *Proc. Infocom 93*, Vol. 3, pp. 1382-91, Mar. 1993.
- [10] T. Hayashi and T. Miyazaki, "High-Speed Table Lookup Engine for IPv6 Longest Prefix Match," *Proc. Globecom*, 1999, vol. 2, pp 1576-1581
- [11] University of Oregon Route Views Project, <http://www.anc.uoregon.edu/route-views/>
- [12] T.C. Chiueh and P. Pradhan, "Cache Memory Design for Network Processors," *Proc. High Performance Computer Architecture*, pp. 409-418, 1999.
- [13] B. Talbot, T. Sherwood, and B. Lin, "IP Caching for Terabit Speed Routers", *Globecom*, 1999.
- [14] P. Newman, G. Minshall, T. Lyon, and L. Huston, "IP switching and gigabit routers," *IEEE Commun. Mag.*, Vol. 35, pp. 64-69, Jan. 1997
- [15] H. Liu, "Reducing Routing Table Size using Ternary-CAM", *Proc. Hot Interconnect 9*, Stanford, 2001
- [16] Merit Networks, Inc., <http://www.merit.edu/ipma>
- [17] V. Srinivasan and G. Varghese, "Fast Address Lookups Using Controlled Prefix Expansion," *ACM Transactions on Computer Systems*, Vol. 17, no. 1, pp. 1-40, Oct. 1999
- [18] B. Lampson, V. Srinivasan and G. Varghese, "IP Lookups Using Multiway and Multicolumn Search", *IEEE Tran. on Networking*, Vol. 7, No. 3, pp324-334, June 1999
- [19] G. Cheung, and S. McCanne, "Optimal Routing Table Design for IP Address Lookups Under Memory Constraints", *Proc. Infocom*, 1999, pp. 1437-44.
- [20] E. Besson, and P. Brown, "Performance Evaluation of Hierarchical Caching in High-Speed Routers", *Proc. Globecom*, 1998, pp. 2640-45
- [21] P. Newman, T. Lyon, and G. Minshall, "Flow Labelled IP: A connectionless Approach to ATM," *Proc. Infocom*, 1996
- [22] Passive Measurement and Analysis project, National Laboratory for Applied Network Research. <http://moat.nlanr.net/pma>