Route Table Partitioning and Load Balancing for Parallel Searching with TCAMs[I]

Dong Lin[1], Yue Zhang[1], Chengchen Hu[1], Bin Liu[1], Xin Zhang[2], and Derek Pao[3]

[1]Dept. of Computer Science and Technology
Tsinghua University, China
lindong05@tsinghua.org.cn
zhang-yue@mails.tsinghua.edu.cn
huc@ieee.org
liub@tsinghua.edu.cn

[2]Dept. of Computer Science
Carnegie Mellon University, USA
xzhang1@cmu.edu

[3]Dept. of Electronic Engineering
City University of Hong Kong, Hong Kong
d.pao@cityu.edu.hk

Abstract

With the continuous advances in optical communications technology, the link transmission speed of Internet backbone has been increasing rapidly. This in turn demands more powerful IP address lookup engine. In this paper, we propose a power-efficient parallel TCAM-based lookup engine with a distributed logical caching scheme for dynamic load-balancing. In order to distribute the lookup requests among multiple TCAM chips, a smart partitioning approach called pre-order splitting divides the route table into multiple sub-tables for parallel processing. Meanwhile, by virtual of the cache-based load balancing scheme with slow-update mechanism, a speedup factor of N-1 can be guaranteed for a system with N (N>2) TCAM chips, even with unbalanced bursty lookup requests.

1. Introduction

IP address lookup is one of the key issues in Internet core routers. Today, an Internet core router, such as the Juniper T640, processes 640G bits per second (Gbps). On condition that each packet is at its minimum size of 40 bytes, such router should complete a packet lookup every 0.5ns. In light of the Moore's law, the growing pace of link transmission speed expects much higher lookup throughput in the near future.

Ternary Content Addressable Memory (TCAM) is a fully associative memory that allows a "don't care" state to be stored in each memory cell in addition to 0's and 1's. It returns the matching result within only one memory access, which is much faster than algorithmic approaches [18] that require multiple memory accesses. As a result, TCAM is widely utilized for the IP lookup nowadays.

Nevertheless, TCAM is not a panacea in that: the memory access speed is slower than SRAM while the power consumption is high. The state-of-the-art 18Mb TCAM can only operate at a speed of up to 266MHz and performs 133 millions lookup per second [4], barely enough to keep up with the 40Gbps line rate today. On the other hand, the size of the route table has been increasing at a rate of about 10-50k entries per year in the past few years [1]. When IPv6 is widely deployed, even more storage space is needed. In pursuit of an ultra-high lookup throughput matching the ever-increasing link transmission speed and the rapidly growing route table, multi-chips system is necessary.

Given multiple TCAM chips in the system, we strive to multiply the lookup rate by using parallel processing techniques with minimal overhead. There are some major concerns in chip-level parallel lookup system design:

1) A partitioning method is needed here to split the entire route table into multiple sub-tables that could be stored in separate TCAMs. It should support dynamic lookup requests distributions, efficient incremental updates, high memory utilization and economical power dissipation.

2) A dynamic load balancing is required for the sake of higher and robust throughput performance in parallel system.

The subject of this paper mainly focuses on the above two issues. The proposed lookup engine is cost-effective and it guarantees a speedup factor of at least *N*-1 by using *N* (*N*>2) TCAM chips, even when the lookup requests can be unevenly distributed. When four TCAMs are used with a partitioning factor of 32, the power consumption is reduced by over 93.75% compared to the naive mode

[I] This work is supported by NSFC (No. 60373007, 60573121 and 60625201), China/Ireland Science and Technology Collaboration Research Fund (CI-2003-02), the Specialized Research Fund for the Doctoral Program of Higher Education of China (No. 20040003048) and the Cultivation Fund of the Key Scientific and Technical Innovation Project, Ministry of Education of China (No. 705003) and Tsinghua Basic Research Foundation (JCpy2005054).
1-4244-0910-1/07/$20.00 ©2007 IEEE.

while the storage redundancy is less than 2%.

The rest of the paper is organized as follows. In Section 2, we describe prior works and some useful observations of the Internet traffic. In Section 3, we propose a table partitioning algorithm which evenly distributes the route table entries among the TCAM chips to ensure high memory utilization. In Section 4, we describe the hardware architecture of the lookup engine and the dynamic load balancing scheme. We present our theoretical performance evaluation in Section 5 and simulation results in Section 6 respectively. Finally, we conclude our work in Section 7.

2. Related Works and Investigation

2.1. Algorithms Based on TCAM

Chip-level parallel TCAMs were deployed to circumvent the limitation of a single TCAM， where issues should be appropriately addressed: i) high memory efficiency, ii) economical power dissipation and iii) balanced traffic load distribution among parallel TCAMs.

Many researchers have strived to optimize the TCAM-based lookup engines with respect to i) and ii): Liu et al. [6] used both pruning and logic minimizing algorithms to reduce the forwarding table sizes, which in turn reduced the memory cost and power consumption of the TCAM. Many other studies [7, 9] developed more power-efficient lookup engines benefiting from a new feature of some TCAMs called "partition-disable". The key idea is to split the entire routing table into multiple sub-tables or *buckets*, where each bucket is laid out over one or more TCAM *blocks*. During a parallel lookup operation, only the block(s) containing the prefixes that match the incoming IP address is (are) triggered instead of all the entries in the original table. In this fashion, TCAM's power consumption can be dramatically reduced. As a key component of such design, generally three kinds of algorithms for partitioning the entire routing table were proposed, i.e., key-ID based [3, 7, 9], prefix trie-based [7] and range-based [8] partitioning. The key-ID approach suffered from uneven sub-table sizes and uncontrolled redundancy, which result in a higher memory and power cost. Trie-based partitioning in [7] can lower the redundancy and unify sub-table sizes, but it required an extra index TCAM to perform the block selection. As a result, two TCAM accesses are occupied for each lookup request. Panigrahy et al. [8] introduced a range-based partitioning framework where they expected to split the routing table into multiple buckets with identical size (number of prefixes). However, the algorithm to determine the partitions which is a vital implemental concern in practice was not presented in [8].

When it comes to iii), few truly effective algorithms have been proposed so far to the best of our knowledge. Without any load balancing mechanisms, eight or more parallel TCAM chips were employed in [8]. But only a speedup factor of five was achieved if the lookup requests were not evenly distributed. A load balancing mechanism was introduced in [9] based on some "pre-selected" redundancy. It assumed that the lookup traffic distribution among IP prefixes can be derived from the traffic traces. In their optimized system evaluation with simulated traffic, a speedup factor of nearly four can be achieved. However, when a certain range of route prefixes, or a certain TCAM block, receives more traffic than others or when the traffic is temporarily or permanently biased to a limited number of route prefixes, multiple selectors frequently try to access the same block. Under such a contentious situation, the system can only fulfill one of the requests. This greatly hampers the average throughput.

2.2. Analysis on Internet traffic

Recall that the performance of a parallel search engine depends heavily on the distribution of lookup requests. Based on the data collected from [10], we observed that the average overall bandwidth utilization was very low but the Internet traffic could be very bursty. In April 2006, the average bandwidth utilization among more than 140 backbone routers was only about 3.1134% and 3.1234% for inbound and outbound traffic respectively[II]. However, workload of individual router can be great bursty during bulk date transfer. Taking router Chin-CERN (40Gbps) as an example, on May 12, 2006, its workload fluctuated between 10Mpkt/s[III] (8%) and 50Mpkt/s (40%), and in the week of 24/04-2006 the workload was increased to over 113Mpkt/s (90%) during peak periods. Jiang et al. [15] suggested that bursts from large TCP flows were the major source of the overall bursty Internet traffic. They identified nine most common causes of source-level IP traffic bursts, one for UDP and eight for TCP flows. Most of these were due to anomalies or auxiliary mechanisms in TCP and applications. It is indicated that TCP's window-based congestion control itself leads to bursty traffic. As long as a TCP flow cannot fill the pipe between the sender and the receiver, bursts always occur. Hence, mapping route table partitions into TCAM chips based on long-term traffic distribution [9] cannot effectively balance the workload of individual TCAM chip during bursty periods.

Instead of mapping route table partitions into TCAM chips relying on long-term traffic statistics, our adaptive load balancing scheme will capture the locality property of Internet traffic using the concept of cache. Locality

[II] As the minimum packet size is 40 bytes, concerning the average packet size of these routers (804 bytes), the average work load of the search engine will be less than 0.16% for both inbound and outbound.

[III] Mpkt/s means million packets per second. When the minimum packet size is 40 bytes, 40Gbps means 125Mpkt/s at maximum.

property of Internet traffic had been widely studied in [11-14]. Analyses on real traces revealed that the conversation[IV] counts were lower than the packet counts, which indicated the existence of the temporal locality. For a real trace, in case of 100-second time interval, top 10% of the highest-activity conversations accounted for over 74% of the packets [11].

3. Smart Route table Partition

In this section, we propose a range-based partitioning algorithm that can evenly allocate the route entries among the TCAM chips to ensure high memory utilization.

To achieve the range-based partitioning, one needs to divide the whole route table into N independent parts by range [8]. Taking IPv4 as an example, an IP address could lay in the space from 0 to (2^{32}-1). We partition this space into N disjoint ranges so that each range maps to about 1/N of the total number of prefixes. We say a prefix falls into a range if any IP address from that prefix can match the range. Formally, a prefix P* falls in a range [a, b] if the range [P0...0, P1...1] intersects with the range [a, b]. To make the route partitioning work well, two points should be addressed here. First, a smart method should be explored to determine the appropriate range boundaries simply. Second, LPM must be guaranteed, because some prefixes may fall into multiple ranges simultaneously.

Our partitioning scheme works in two steps. First, a routing trie is constructed using the route table. LC-trie [17] can be certainly used here to save memory space, but for the sake of clear description we use 1-bit trie for demonstration. (Figure 1 is an example of 1-bit trie built from a route table.) Second, sub-trees or collections of the 1-bit trie are successively carved out and mapped into an individual routing group.

3.1. Pre-order Splitting

Here is the description of the trie-based partitioning algorithm called pre-order splitting. (Figure 2 is the pseudo-code of this algorithm.) This algorithm takes a parameter b that denotes the number of TCAM buckets as input. The output is a set of b sub-tables with equal size and a set of (b-1) *reference nodes* which are depicted in Figure 2. During the partitioning step, the entire trie is traversed in pre-order looking for a prefix node. Every time a prefix

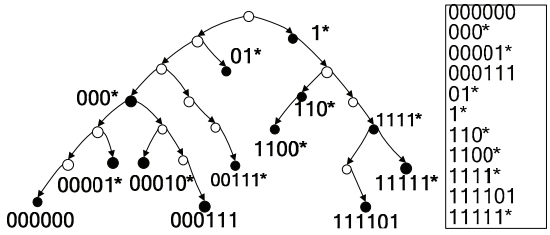
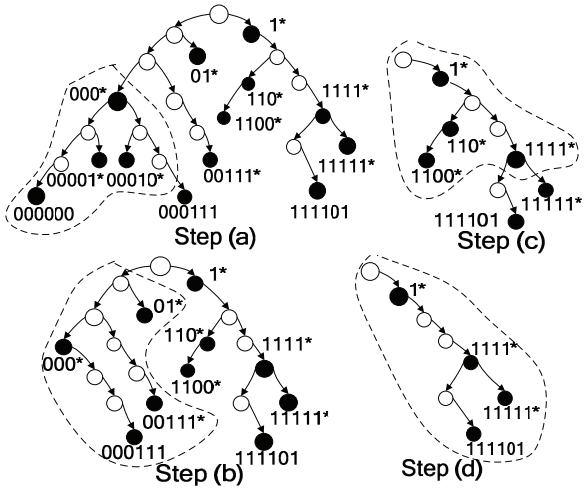


Figure 1. An example route table and the corresponding 1-bit trie built from it. [V]

```
Preorder-split (b)
{
   m=count/b; j=b; u=1;
   while (j! =1){
      p=root;
      for (i =0; i<=m; i++){
         p=next prefix node in pre-order;
         push(p);
         put prefix(p) in bucket[u];
      }
      p=next prefix node in pre-order;
      save prefix(p) as the reference_node[u];
      u++;
      while (stack is not empty){
         p=pop;
         while（node p has no children node）{
            delete(p);
            p=parent(p);
         }
      }
      j--;
   }
   save the rest prefix nodes as bucket[b];
}
```

Figure 2. Pre-order splitting[VI].



No.	Bucket Prefixes	Ref. Node	Range Low	Range High
1	000*,000000, 00001*,00010*	000111	000000	000110
2	000*,000111, 00111*,01*	1*	000111	011111
3	1*,110*, 1100*,1111*	111101	100000	111100
4	1*,1111*, 111101,11111*		111101	111111

Figure 3. Four iterations of the pre-order splitting algorithm.

[IV] A conversation was defined as a sequence of packets toward the same destination address within a fixed time interval [11].

[V] The prefixes of only the black nodes are in the route table. A "*" in the prefixes denotes the start of don't-care bits.

[VI] Here *count* is the number of prefixes in the 1-bit trie, *root* is the root node of the 1-bit trie, *prefix (p)* is the prefix of node *p*, *parent (p)* is the parent node of *p* in the 1-bit trie, *push (p)* means to save *p* into stack, *pop* means to load an element from stack, and *delete (p)* means to delete node *p* from 1-bit trie.

node is encountered, it will be saved in a TCAM bucket and pushed into a stack at same time. When the program gets an average number of prefix nodes, it will pop an element from the stack and delete the corresponding prefix node (remount to its parent node) which has no child nodes from the entire trie. This step will continue until the stack is empty. After this step, some parts of the tire are carved out. The program will get a sub-table with equal size fitting to a TCAM bucket in each loop. Finally, the rest of the prefix nodes in the last loop will be saved in the last TCAM bucket. During the whole process, the *reference nodes* will be saved too. Figure 3 instantiates how prefix nodes are divided into four buckets from the 1-bit trie depicted in Figure 1. It is obvious that the first TCAM bucket's low range point must be (0…0) and the last TCAM bucket's high range point must be (1…1). We will use the *reference nodes* to calculate the other boundary points of each TCAM bucket. For instance, P* is the *reference node* of TCAM bucket u (not the last one). So its high range point will be (P0…0 - 1) while the TCAM bucket [u+1]'s low range point will be (P0…0). In this way, we have b TCAM buckets and each of them has a pair of boundary points.

To lookup the route for an input address a, only the TCAM bucket [u] is activated, where the value of a lies within the low and high range of bucket [u]. To ensure correct lookup result, prefixes lying along the boundary of 2 adjacent buckets are duplicated in the 2 buckets concerned automatically. Furthermore, the default prefix (0.0.0.0/0) will be copied to all the buckets initially. We refer to this prefix duplication as the redundancy. As shown in Figure 1, level of prefix nesting in the routing table is no more than 6, i.e. there are at most 6 valid prefixes along a path from the root to a leaf in the 1-bit trie. Hence, the amount of redundancy is no more than 6*b. In IPv4, length of prefixes can be between 8 to 32, whereas in IPv6 length of prefixes can be between 16 to 64, and 128 [5]. If there are b buckets in the system, then the redundancy is no more than 24*b for IPv4, and no more than 50*b for IPv6.

The prefix update is very simple. During an insertion operation, a prefix P will be compared with the boundary points and be copied to the corresponding buckets. During a deleting operation, there will be no extra operation for a non-boundary prefix. For a boundary prefix, the new ranges of its TCAM bucket should be re-calculated. Since the boundary points are only determined by the *reference node*, we just need to find a new *reference node*. It can be determined by a simple pre-order searching in the 1-bit trie that costs only a few clock cycles. As shown in Figure 1, when the prefix node 1* is deleted, the new *reference node* for bucket No.2 will be 110* (the next prefix node of 01* in pre-order). As a result, the range for both bucket No.2 and No.3 will be changed into (000111, 101111) and (110000, 111100), respectively. Another advantage is that it is not difficult to balance the size of two adjacent TCAM

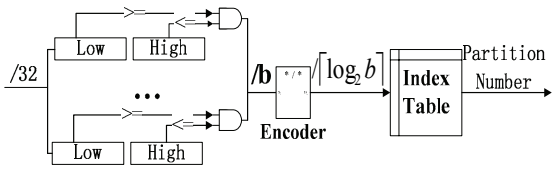


Figure 4. Schematics of the indexing logic.

Bucket	Range Low	Range High	Size
1	0.0.0.0	32.113.255.255	5807
2	32.114.0.0	62.101.191.255	5807
3	62.101.192.0	64.76.75.255	5807
4	64.76.76.0	65.164.213.255	5807
5	65.164.214.0	66.187.243.255	5807
6	66.187.244.0	68.166.255.255	5807
7	68.167.0.0	70.169.57.255	5807
…			
30	213.182.109.0	216.158.137.255	5807
31	216.158.138.0	218.103.47.255	5807
32	218.103.48.0	255.255.255.255	5110

Figure 5. The detail result with b=32 on Equinix.

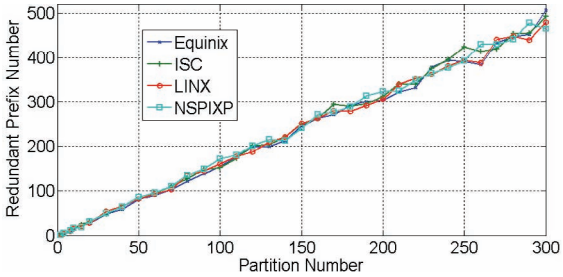


Figure 6. Partition redundancy.

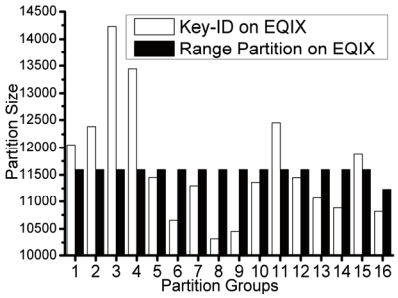


Figure 7. Comparasion on key-ID and range partition.

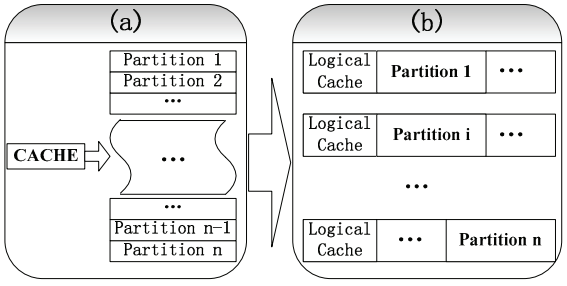


Figure 8. Cache organizations: (a) centralized cache; (b) distributed caches

buckets dynamically.

Figure 4 illustrates the pipelined structure of the Indexing Logic for TCAM block selection. It is composed of pairs of parallel comparing logics and an index table. Each pair of parallel comparing logic corresponds to one TCAM bucket and is composed of two registers which store the boundary points of each TCAM bucket. Next to the parallel comparing logics is an index table with encoder. The index table which stores the bucket distribution information returns a partition number indicating which bucket may contain the prefix matching the IP address by the encoded information. Because the data width of the Indexing Logic is fixed and only simple "compare" operation is executed, it can work at a very high speed. In this paper, the Indexing Logic has been set as the entry of the entire parallel system.

3.2. Evaluate the Pre-order Splitting

We have analyzed the route tables [VII] provided by route-views [2]. We chose different b for test. In order to prevent the large size of the last bucket, we chose $\lceil count/b + 24 \rceil$ as the average size instead of $\lceil count/b \rceil$. Figure 5 shows some detail results partitioned on Equinix with b=32 and the redundancy of the entire system is only 51. Figure 6 shows the relationship between bucket number b and the whole redundancy among four route tables. The redundancy for all route tables is less than $2b$ and it is much less than the theoretical worst-case estimation ($24b$).

We also have a comparison between our range-based partitioning method and the key-ID approach [9]. Since the key-ID approach can only based on 10-13bit (which is the best configuration for both redundancy and even sub-table size), we chose b as 16 and chose $\lceil count/16 + 24 \rceil$ as the average size of each bucket. Figure 7 shows that besides more redundancy, the key-ID approach suffers from uneven sub-table sizes seriously while our partitioning method can guarantee precisely even size of each bucket and little redundancy.

4. Logical Cache for Load Balancing

4.1. Cache Organization

As mentioned in Section 2.2, temporal locality of the Internet traffic is much stronger in the core routers due to the greater effect of heavy flow aggregations. To achieve higher lookup throughput, a straightforward design with cache is to deploy a first stage caches working in front of a second stage data TCAMs. An obvious drawback of this conventional approach is that the cache is required to operate at N times the speed of TCAM if there are N parallel TCAMs in the system, which is impractical.

As we have mentioned in Section 1, TCAM vendors have started providing mechanisms called TCAM blocks. Since there are multiple TCAM chips, we can use some blocks to create *logical caches*. In this case, the commercial TCAMs can be competent for supporting such a cache mechanism. No additional cache module implies fewer pins and less packaging cost. Furthermore, employing the existing TCAM cells as logical caches exhibits a better performance cost ratio. So we propose a parallel system with distributed logical caches as Figure 8(b) depicts.

Based on our partitioning algorithm, we partition each TCAM into small buckets with a size of $\lceil count/b + 24 \rceil$ so that each bucket can store in one single partition. Now we use the partition as a basic unit to allocate the prefixes among the TCAMs. We also select some blocks from each TCAM to serve as logical caches. Suppose there are 4 TCAMs and each one has 8 partitions plus 1 logical cache. Therefore, there are totally 32 partitions, so b equals to 32.

4.2. Logical Caches for Load Balancing

The detailed implementation architecture of the parallel lookup engine is presented in Figure 9. Given an incoming IP packet to be searched, the IP address is extracted and delivered to the Indexing Logic (see Figure 4) for a comparing operation. The Indexing Logic will return a partition number indicating the "home" TCAM that may contain the matching prefixes. The Adaptive Load Balance Logic sends the IP address to the home TCAM or an alternate TCAM for processing based on the length of the FIFO queues. A feed back logic is also settled to operate the cache-missed packets. Since multiple input queues and feeding back exist in the proposed scheme, the incoming IP addresses can be processed in a non-FIFO order. The Re-ordering logic maintains the original sequence by using the time stamp attached.

The adaptive load balance logic distributes a new lookup request to the TCAM chip with the shortest input queue. With the feedback mechanism depicted in Figure 9, there are three different alternatives. 1) If the incoming IP address has been sent to its home TCAM, it will get a search operation on the partition indicated by the Indexing Logic and the final result is done. 2) If the incoming IP address has been sent to a non-home TCAM, it will get a search operation on the logical cache. When it is cache-matched, the result is guaranteed by the RRC-ME, so the final result is done. 3) The logical cache of the TCAMs only holds a small number of prefixes. When a cache miss occurs, it must be sent back to the home TCAM via the feedback to the corresponding FIFO and

[VII] They are Equinix located in Ashburn, VA on 2006-05-02 00:42(UTC) with a size of 185076; ISC (PAIX) located in Palo Alto CA, USA on 2006-05-02 00:14(UTC) with a size of 186843; LINX located in London, GB on 2006-05-02 00:36(UTC) with a size of 186582; DIXIE (NSPIXP) located in Tokyo, Japan on 2006-05-02 00:17(UTC) with a size of 188526.

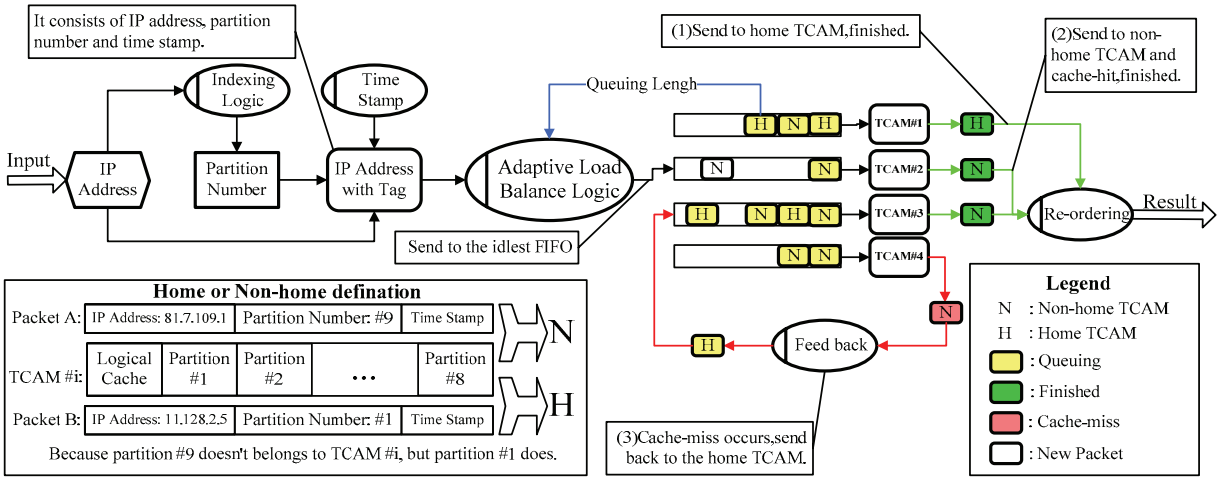


Figure 9. Schematics of the complete implementation architecture.

case 1) happens again.

4.3. Cache IPs or Cache Prefixes

Generally two kinds of algorithms for cache mechanism are proposed in route loopup literatures, caching destination addresses (IPs) [11-13] and caching prefixes [14]. Binary CAM can be used if IPs are cached. However, using BCAMs imposes a much higher overhead in the proposed system. The second disadvantage of caching IPs is that a much larger cache size is required. A number of papers [11-13] have demonstrated that the cache hit rate can be over 90% with 8,192 cache entries by caching IPs, while the cache hit rate can easily achieve 96.2% with 256 cache entries by caching prefixes [14].

For caching prefix, one important issue is worth being mentioned here. Suppose p and q are two prefixes where q is a subrange of p, i.e. p is a prefix of q. If there is a lookup request r redirected to the cache and p is the LMP of r, then p will be added to the cache. However, in some later time if another request r' (whose LMP is q) is redirected to the cache, then the cache will return p as the lookup result (where the correct result should be q). To resolve this problem, we adopt the RRC-ME algorithm [14] in managing the cache. In the above example, the shorter prefix p will be expanded into a longer prefix, based on the *Minimal Expansion Prefix* (MEP) method in [14]. For instance, there are only two prefixes in a route table, i.e. 1* and 111*. Hence, the MEP for request 1111 is 111*, the MEP for request 1000 is 10*(it has the same next hop as 1*), the MEP for request 1100 is 110*(it has the same next hop as 1*). Since the expanded prefixes are disjoint, there is one and only one possible match result for every input address. Thus, the match result, if any, returned by looking up a cache is valid. Another advantage of prefix expansion method is that any update to the cache block only requires one TCAM cycle because the prefixes need not be ordered.

4.4. Slow-update

In the proposed logical cache organization, a TCAM chip is not available to perform IP lookup during cache updates. Hence, a high cache update frequency will seriously affect the system performance. Moreover, immediate cache update in the event of cache miss is very difficult since the system has to evaluate the MEP decomposition of the corresponding prefix. We shall show that a slow-update mechanism with LRU algorithm can achieve nearly the same cache-hit rate as immediate update when the cache size is large enough.

Slow-update mechanism can be considered as a sampling update. Only one cache-missed element is updated within a predefined interval, i.e. D TCAM cycles. During this period, the other cache-missed elements are ignored. For instance, a cache-missed element is detected at time t and the cache update module is free, then the MEP of this cache-missed element will be updated to the cache at time $t+D$. The other cache-missed elements detected during time $t+1$ to $t+D$ are ignored. Since the slow-update mechanism is very easy to implement, a commercial CPU is competent for the processing.

We have evaluated our slow-update mechanism with traces. Unlike the "pretreated"[VIII] traces published on [16], we collect un-pretreated and representative traffic traces from one backbone router of the China Education and Research Network (CERNet[IX]).

The monitoring router is located in the network center of Tsinghua University which belongs to Beijing Regional

[VIII] Due to the commercial secret, both the destination and source IP addresses have been changed. Most of them are 10.*.*.*.

[IX] CERNet is one of the biggest networks in China with over 18 millions users. There are over 1300 schools, research or government institutions among 31 provinces connected to CERNet via 8 regional network centers.

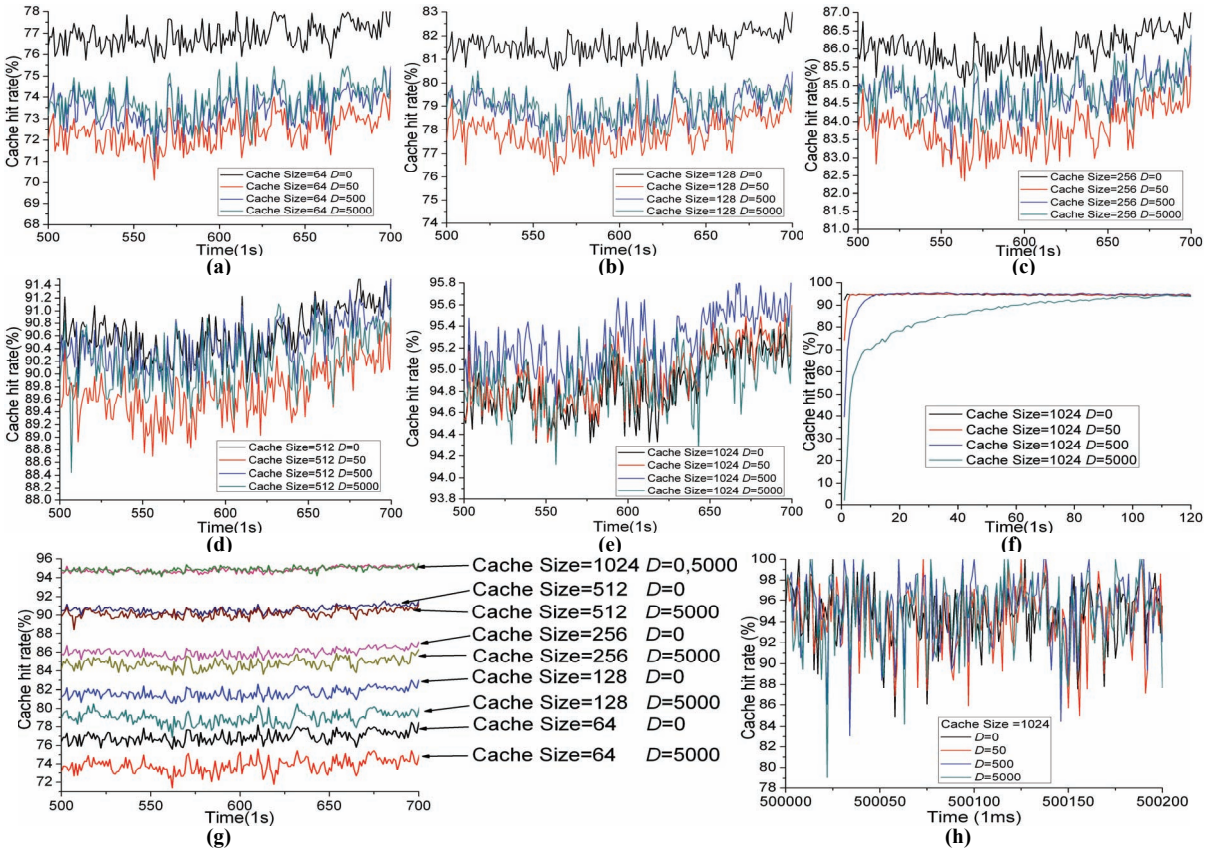


Figure 10. Cache hit-rate of Slow-update policies.

Network Center. It operates at 1Gbps of the Ethernet link bandwidth. The trace is collected from 10:15 to 10:30 on Sep. 19, 2006 and only outbound packets were recorded.

As mentioned in Section 2.2 that the average overall bandwidth utilization is very low but the Internet traffic could be very bursty. In order to simulate the most serious situation, the trace is first sorted according to the packet arrival timestamp to ensure each packet appears in the correct time order. Then it is changed into back-to-back mode but we still measure the statistic per second by the timestamps. This kind of treatment deteriorates the experimental environment and the experimental result can be well guaranteed even in the other serious situations. Then only the information in the middle 200 seconds (500 to 700) of this 15-minute trace is reserved to avoid the possible warm-up and trail interference. The route table (Equinix) used for this experiment was borrowed from [2] and contains 185,076 entries.

Figure 10 shows the detail results. We find that the update-delay does not affect so much if the cache size is large. Figure 10 (a) demonstrates that when the cache size is 64, the immediate update (D=0) outperforms the slow-update approach, the cache hit rate is about 5% higher than the others. Figure 10 (b) draws the results of caching 128 prefixes. Though the immediate update still outperforms than the others (the cache hit rate is about 3% higher than the others), the cache hit rate line with D=50 is even worse than the D=5000's one. When the cache size is 256 (Figure 10 (c)), the immediate update's cache hit rate is only 1.5% higher then the others. We go on increasing the cache size. When there are 512 cache entries (Figure 10 (d)) and D is 5000, the cache hit-rate can still achieve over 90%. In this case it is truly hard to distinguish them (immediate update and slow-update) clearly. When the cache size is increased to 1024 (Figure 10 (e)), the slow-update mechanism may even outperform the immediate update, the cache hit rate is also increased to 95%. For the sake of clear illustration, we put Figure 10 (a)(b)(c)(d)(e) together. As shown in Figure 10 (g), the cache hit rate with different D is getting closer as the cache size increases. So we conclude that: 1)It is hard to say that bigger or smaller D is absolutely good or not for the performance when the cache is quite large. 2)As the cache size is increased, impact of D becomes less significant. 3)By increasing the cache size, we can achieve a higher cache hit rate while the system can also tolerate a bigger D.

We acknowledge that the slow-update mechanism has its drawbacks. First, it takes a long time for the system to reach steady state in a cold start. As shown in Figure 10(f), when the cache size is 1024 and D equals 5000, the cache needs about 100 seconds to warm-up and this is obviously

unacceptable. But we can suggest some strategy to resolve this problem. For example, we can use a small D value at the beginning to achieve a reasonable cache hit rate quickly. As shown in Figure 10(f), when D equals to 50, the cache can be full filled in one second. After the warm-up period, D could be set back to a big value again. Since we cache prefixes not IPs, the un-updated prefixes may still have a great chance to be matched by the other flows. And some prefixes that are frequently accessed can stay in the cache by LRU. By combining LRU with slow-update mechanism, a nice performance can be achieved when the cache size is large enough.

Second, slow-update mechanism leads to much fluctuation. Figure 10(h) is an amplificatory demonstration of Figure 10(e), the statistic interval has been reduced to 1 ms. We find that the cache hit-rate fluctuates seriously. The cache hit rate may even drop to 79% at 500022 ms when D equals 5000. Generally speaking, larger D leads to more fluctuations. Hence, bigger buffers are needed for the slow-update mechanism.

5. Performance Analysis

5.1. Lower Bound of the System Performance

As mentioned above in Section 4, a practical parallel lookup system should be able to handle bursty traffic efficiently. We discuss the lower bound of the system performance in this section. Because the update cost of proposed system is only $1/D$, when D is larger enough, i.e. 5000, the update cost is only 0.02%. Thus, we shall neglect the update cost in the analysis.

For the sake of clear description, let us consider the following scenario. There are N (N>2) TCAMs with the aforementioned structure. Suppose the maximum bandwidth of one TCAM is MW_{max} (M stands for the speedup factor of TCAM), and the average hit rate of cache is x. In a practical parallel system, the most serious situation would be that all the workload has been sent to a single TCAM. As a result, the performance of entire system would be much worse than the theoretical maximum. Therefore, we shall address such a problem that in order to guarantee NW_{max} throughput in a system with N TCAMs, what kind of equation (concerning x and M) should be hold true.

In the case above, with the operation of load balancing, the other N-1 TCAMs will work as logical caches. And the TCAM which supposes to finish all the workload itself becomes "second stage" spontaneously. It just needs to operate the cache-missed packets from the other N-1 TCAMs. In the stable state with finite buffer, the following equation should hold true.

$$(N\text{-}1)(MW_{\max}) \geq NW_{\max} \quad (1)$$

$$NW_{\max}(1-x) \leq MW_{\max} \quad (2)$$

Thus,

$$M \geq \frac{N}{N\text{-}1} \quad (3)$$

$$1-x \leq \frac{M}{N} \quad (4)$$

Since M stands for the speedup factor of each TCAM, we choose its minimal value. Therefore, we could further derive the average cache hit rate x as follows.

$$x \geq \frac{N-2}{N-1} \quad (5)$$

Further, we can obtain the system speedup S from bandwidth utilization as follows.

$$S = \frac{NW_{\max}}{NMW_{\max}} N = \frac{N}{M} \geq N-1 \quad (6)$$

From (3)(5)(6), we find that M can be decreased as the growing N. But x should be increased. For illustration, when N=4, M and x should be 4/3 and 2/3 at least. Hence, S could be 3. When N=8, M and x should be 8/7 and 6/7 respectively. Here, S could be 7. In such cases, the system can tackle NW_{max} throughput in despite of the bursty traffic.

5.2. Buffer & Power Consumption

In previous part, a clear scenario shows the result of load balancing while the input traffic is very bursty. The over-loaded cache-missed packets due to the slow-update mechanism can also be identified as a special case of burst traffic. As a result, x depends on not only the cache size but also the length of buffer. For example, there will be no packet drop when x measured with a statistical interval L is higher than (N-2)/(N-1) at any time.

Furthermore, the length of buffer also greatly concerns the total delayed time in our system which covers both reordering and queuing. Since there is no blocking or Round-Robin mechanism in our system, any packet will be queued for twice at most. In that case, both the queuing and reordering would cost 2L at most (L denotes the buffer length of each TCAMs). Our aim is to find a proper L for the system to achieve a low loss rate, i.e. 10^{-8}. Since L depends on the characters of traffic, we may only give an empirical value by experiments in Section 6.

As we know that the main power consumption for TCAM depends on the entries triggered during a parallel search processing, assume the power consumption for a TCAM without partitioning search is P. Since we adopt b partitioning search zones in TCAM, and the average lookup times is no more than 2, so the power consumption should be less than $2P/b$[X]. When b equals to 32, the power

[X] When b is increased to a larger value, i.e. 200, the partition size will be less than 1K which is smaller than the logical cache. As a result, the power consumption will be less than $2000P/Z$ where Z denotes the size of route table.

consumption can be reduced by 93.75% at least.

6. Experiments and Simulations

In addition to the theoretical analysis we have run a series of experiments and simulations to measure the algorithm's performance and adaptability on the most serious traffic load distributions. Here we choose M=4/3, N=4, D=5000 and b=32 for demonstration.

6.1. Experimental Scheme

We insist that the parallel system must be applicable to uneven workload distribution caused by the bursty traffic. Hence, the experiment herein mainly focuses on bursty traffic. Traces collected from the real world do have some bursty traffic, but as the low average traffic (stated in Section 2.2), we will not be able to test a parallel system with a strong and continuous input. In order to test the proposed structure and get the buffer length, the following steps have been taken.

1) We overlap four traces [XI] to generate a new 15-minute trace. It contains $2.436*10^8$ packets and the average active connections (measured in 64 second time out) are over 400K.

2) After the step 1), the timestamp of this new trace is ignored and this new trace is changed into a back-to-back mode. In order to guarantee 100% input, we just need to ensure the proportion of the maximal input traffic (NW_{max}) to the maximal throughput of all the TCAMs ($N*MW_{max}$) is 1:M. Suppose N is equal to 4, then M should be 4/3 by the theoretical analysis presented in Section 5. It means that each TCAM can serve for one packet every 4 clocks (every 4 clocks, totally 4 packets are finished by 4 TCAMs) while there are 3 packets arrived in 4 clocks at most. Analysis shows that the actual utilization of each TCAM bucket is greatly different. Some of the TCAM buckets carry the most of the workload. As shown in Figure 11, the top five TCAM buckets account for over 85% of the total workload. This character can be used to construct an extremely uneven bucket distribution for the system.

3) By introducing some "pre-selected" redundancy [9] to our system, an even workload can be easily constructed and the system performance can be greatly increased. But this kind of operation requires that the lookup traffic distribution among IP prefixes can be derived in real time and the traffic distribution should be quite stable which has been proved impractical in practice due to the busty traffic in Internet. So it is impossible to "optimize" the partition organization (bucket distribution) which means the system could face the most serious situation (most of

[XI] They were collected at 10:15 – 10:30, 14:55 – 15:05, 17:58 – 18:13 and 21:52 – 22:07 on 2006-09-19 respectively.

Bucket	Range Low	Range High	Proportion
32	218.103.48.0	255.255.255.255	33.0951%
28	209.215.80.0	211.144.215.255	17.0767%
2	32.114.0.0	62.101.191.255	14.2720%
21	202.56.193.0	202.169.219.255	11.7201%
22	202.169.220.0	203.101.67.255	8.24875%
13	161.222.160.0	192.44.143.255	2.88938%
31	216.158.138.0	218.103.47.255	2.88427%
20	200.150.240.0	202.56.192.255	1.84324%
12	144.243.208.0	161.222.159.255	1.65799%
10	85.120.78.0	130.117.255.255	1.62507%
11	130.118.0.0	144.243.207.255	0.83727%
29	211.144.216.0	213.182.108.255	0.76708%
9	81.7.109.0	85.120.77.255	0.63909%
…			
14	192.44.144.0	193.5.115.255	0.02988%
18	198.179.209.0	200.3.213.255	0.01591%
24	204.27.185.0	205.172.15.255	0.00934%

Figure 11. Workload of different TCAM buckets.

TCAM No.	Bucket No.	Total Traffic
1	32,28,2,21,22,13,31,20	92.030%
2	1,3,4,5,6,7,8,9	2.0902%
3	10,11,12,14,15,16,17,18	4.3540%
4	19,23,24,25,26,27,29,30	1.5263%

Figure 12. Mapping of buckets and workload of TCAM chips.

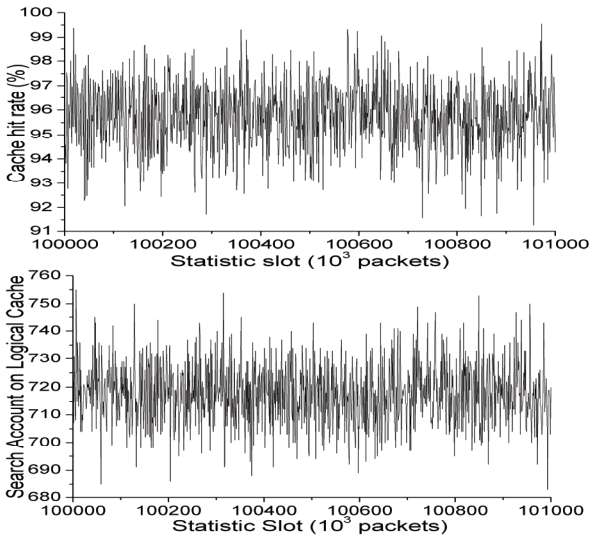


Figure 13. Result on bursty traffic.

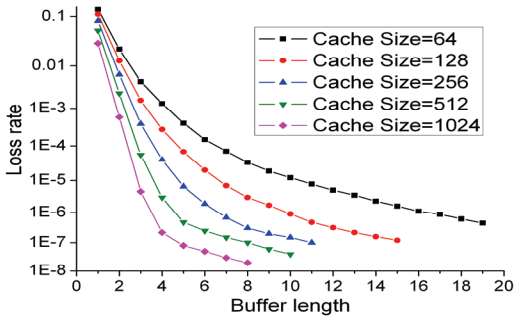


Figure 14. Loss rate and buffer length.

the traffic has been sent to a single TCAM) sometimes. In this paper, the most serious situation is created in the experiment to demonstrate the great load balancing of proposed solution. The partition organization on four TCAMs has been shown in Figure 12. The TCAM No.1 accounts for 92.03% of the total traffic which means the parallel system suffers from uneven workload seriously.

6.2. Simulation Result

Now, we use the trace generated by step 2) to test the system organized by step 3), the cache size has been set to 1024 and D=5000. Simulations show that the system can easily achieve 100% throughput with very low power consumption. Figure 13 shows part of the result, the statistic interval is 10^3 packets. It is shown that the cache hit rate still achieve 95% in the parallel system while the average search account on logical caches is over $0.7*10^3$.

With the experiment scheme above, we also test the proposed system with D=5000 and different cache sizes (64, 128, 256, 512 and 1024). To avoid the warm-up interference, the first 10^7 packets has been ignored. As shown in Figure 14, the buffer length is also decreased as the increasing cache size. When the cache size is 64, a longer buffer (17) is needed to achieve a loss rate about 10^{-6}, while the one with a cache size of 128 can reduce the buffer length by 6. By increasing the cache size, the smaller buffer can be achieved. When the cache size is 1024, a loss rate about 10^{-8} can be easily achieved with a buffer length about 8 in despite of the bursty traffic.

7. Conclusion

Increasing the lookup throughput and reducing the power consumption of the TCAM-based lookup engine while keeping economic memory storage utilization are the three primary issues in this paper. We first give a simple but efficient TCAM table partitioning method. It supports incremental updates efficiently with little redundancy and is easily scalable to IPv6. Motivated by the analysis on real-life Internet traffic, we then devised an adaptive load balance scheme with logical cache to solve the bursty traffic. Meanwhile, the update mechanism used for the logical cache is very simple and efficient which is greatly different from the traditional ones. Both the partitioning method and the logical cache with slow-update mechanism are quite flexible. They can be easily modified for different requirements, such as bigger cache, longer update-delay, more TCAM buckets and so on. Given 2% more memory space, the proposed scheme increases the lookup throughput by a factor of three when a quaternary TCAMs' structure is implemented and significantly cuts down the power consumption.

8. Acknowledgment

The authors would like to express their appreciations to Dr. Huan Liu of Stanford University for his constructive suggestions on this paper and the DRAGON-Lab (Distributed Research & Academic Gigabits Open Network Lab) of Tsinghua University for providing the real traces.

References

[1] Huston G. Route Table Analysis Reports (March 2006). http://bgp.potaroo.net/.
[2] Route Views. http://routeviews.org/.
[3] Xin Zhang, Bin Liu, Wei Li, Ying Xi, David Bermingham and Xiaojun Wang, "IPv6-oriented 4*OC768 Packet Classification with Deriving-Merging Partition and Field-Variable Encoding Algorithm", INFOCOM2006, 23-29 April, Barcelona, Spain.
[4] CYRESS. http://www.cypress.com/.
[5] Y. K. Li, D. Pao, "Address Lookup Algorithms for IPv6", IEE Proceedings-Communications, Vol. 153, No. 6, pp. 909-918, Dec. 2006.
[6] H. Liu, "Route table Compaction in Ternary CAM", IEEE Micro, 22(1):58-64, January-February 2002.
[7] F. Zane, G. Narlikar, A. Basu, "CoolCAMs: Power-Efficient TCAMs for Forwarding Engines", INFOCOM2003,San Francisco.
[8] R. Panigrahy, S. Sharma, "Reducing TCAM Power Consumption and Increasing Throughput", Proceedings of HotI 2002, Stanford, California, USA.
[9] Kai Zheng, Chengchen Hu, Hongbin Liu, Bin Liu,"An ultra-high throughput and power efficient TCAM-based IP lookup engine", INFOCOM2004, Hong Kong, China.
[10] Abilene. http://www.abilene.iu.edu/noc.html.
[11] Woei-Luen Shyu, Cheng-Shong Wu, and Ting-Chao Hou. "Efficiency Analyses on Routing Cache Replacement Algorithms", ICC'2002, 28 April - 2 May, New York City, NY, USA.
[12] Tzi-cker Chiueh,Prashant Pradhan, "High-Performance IP Route table Lookup Using CPU Caching", INFOCOM'99, March 23, New York City, NY, USA.
[13] Bryan Talbot, Timothy Sherwood, Bill Lin, "IP CACHING FOR TERABIT SPEED ROUTERS", GLOBECOM '99, 8 DEC, Rio de Janeiro, Brazil.
[14] Mohammad J. Akhbarizadeh and Mehrdad Nourani, "Efficient Prefix Cache for Network Processors", High Performance Interconnects 2004, 23 – 25 August, Stanford University, USA.
[15] H. Jiang , C. Dovrolis, "The effect of flow capacities on the burstiness of aggregated traffic". PAM'04, April 2004, Antibes Juan-les-Pins, France.
[16] Passive Measurement and Analysis (PMA) Project. http://pma.nlanr.net/
[17] S. Nilsson and G. Karlsson, "IP-Address Lookup Using LC-Tries", IEEE Journal on Selected Areas in Communications, Vol. 17, No. 6, pp. 1083-1092, June 1999.
[18] M. A. Ruiz-Sanchez, E. W. Biersack and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms", IEEE Network, pp. 8-23, March/April 2001.