2012 32nd IEEE International Conference on Distributed Computing Systems

CLUE: Achieving Fast Update over Compressed Table for Parallel Lookup with Reduced Dynamic Redundancy

Tong Yang, Ruian Duan, Jianyuan Lu, Shenjiang Zhang, Huichen Dai and Bin Liu*
Tsinghua National Laboratory for Information Science and Technology
Department of Computer Science and Technology, Tsinghua University, Beijing, China

Abstract— **The sizes of routing table in backbone routers continue to keep a rapid growth and some of them currently increase up to 400K entries [1]. An effective solution to deflate the large table is the routing table compression. Meanwhile, there is an increasingly urgent demand for fast routing update mainly due to the change of network topology and new emerging Internet functionalities. Furthermore, the Internet link transmission speed has scaled up to 100Gbps commercially and towards 400Gbps Ethernet for laboratory experiments, resulting in a raring need of ultra-fast routing lookup. To achieve high performance, backbone routers must gracefully handle the three issues simultaneously: routing table *C*ompression, fast routing *L*ookup, and fast incremental *U*pdat*E* (CLUE), while previous works often only concentrate on one of the three dimensions.**

To address these issues, we propose a complete set of solutions—CLUE, by improving previous works and adding a novel incremental update mechanism. CLUE consists of three parts: a routing table compression algorithm, an improved parallel lookup mechanism, and a new fast incremental update mechanism. The routing table compression algorithm is based on ONRTC algorithm [2], a base for fast TCAM parallel lookup and fast update of TCAM. The second part is the improvement of the logical caching scheme for dynamic load balancing parallel lookup mechanism [3]. The third one is the conjunction of the trie, TCAM and redundant prefixes update algorithm. We analyze the performance of CLUE by mathematical proof, and draw the conclusion that speedup factor is proportional to the hit rate of redundant prefixes in the worst case, which is also confirmed by experimental results. Large-scale experimental results show that, compared with the mechanism in [3], CLUE only needs about 71% TCAM entries, 4.29% update time, and 3/4 dynamic redundant prefixes for the same throughput when using four TCAMs. In addition, CLUE has another advantage over the mechanism in [3] -- the frequent interactions between control plane and data plane caused by redundant prefixes update can be avoided.

I. INTRODUCTION

Internet has maintained a rapid growth for more than ten years, which brings three major issues: **i)** ***routing table compression*** -- due to an annual increase of about 15% of the routing table size [4], ISPs struggle to suppress the table growth, so as to further postpone the requirement for upgrading the router's memory; **ii)** ***routing lookup*** -- to handle the current tens of gigabit-per-second traffic, the backbone routers must be able to forward hundreds of millions of packets per second, thereby bringing huge pressure to routing lookup; **iii)** ***fast update*** -- facing more and more frequent routing updates, routing table must be incrementally updated as fast as possible. These three issues of ***C***ompression, ***L***ookup, and ***U***pdat***E*** are abbreviated to **CLUE**, and CLUE also stands for a set of solutions for them in this paper.

With regard to **i)**, the typical solutions are [5-8]. Although they can achieve high compression ratio, they fail to achieve fast updates when Ternary Content Addressable Memory (TCAM) is used. Therefore, we proposed ONRTC algorithm in [2], which supports parallel lookup and fast update.

With regard to **ii)**, TCAM-based solutions are usually adopted in backbone routers. TCAMs are fully associative memories that allow a "don't care" state to be stored in each memory cell in addition to *0*s and *1*s. One TCAM access can finish one routing lookup operation while software-based solutions might need multiple memory accesses. Therefore, TCAM-based solutions are much faster and widely used for the routing lookup nowadays. However, the TCAM-based solutions are of high power consumption and cost. In order to achieve power efficiency, F. Zane et al. proposed a bit-selection architecture [9], which hashes a subset of the destination address bits to a TCAM partition, thus making TCAM-based routing table power efficient. This algorithm is referred to as ID-bit partition algorithm in this paper. It only concentrates on reducing power consumption, but cannot accelerate routing lookup.

Take an Ethernet link with 400Gbps for example, suppose each packet is at its minimum size of 64 bytes, routers should complete a packet lookup every 1.28ns theoretically, while the common TCAM runs at around 166 MHz. Therefore, parallel lookup with multiple TCAMs is destined if TCAM solution is adopted. Towards this target, previous works have proposed two mechanisms: Kai's algorithm [10] and Dong's algorithm [3].

In [10], Kai Zheng et al. proposed a TCAM-based parallel architecture, which employs an intelligent partitioning algorithm, takes advantage of the inherent characteristics of Internet traffic, and increases packet forwarding rate multiple times over traditional TCAMs. Power consumption is also reduced by ID-bit partition algorithm. In order to achieve load balancing, Kai Zheng et al. added 25% redundant prefixes to the system based on the long period statistical results. For convenience, this algorithm is referred to as Statistical Load balancing Parallel Lookup (SLPL) in this paper.

However, according to the data mining results in [3], the average overall bandwidth utilization was low but the Internet traffic could be very bursty. Hence, during bursty

*Corresponding author: liub@tsinghua.edu.cn.
Others: {yang-t10, dra08, lu-jy11, zsj09, dhc10}@mails.tsinghua.edu.cn.
Supported by NSFC (61073171, 60873250), Tsinghua University Initiative Scientific Research Program, the Specialized Research Fund for the Doctoral Program of Higher Education of China(20100002110051).

1063-6927/12 $26.00 © 2012 IEEE
DOI 10.1109/ICDCS.2012.79

periods, mapping routing table partitions into TCAM chips based on long-term traffic distribution observations cannot balance the workload of individual TCAM chip effectively. Therefore, the redundant prefixes should be dynamic adjusting, thus Dong Lin et al. proposed a power-efficient parallel TCAM-based lookup engine with a distributed logical Caching scheme for dynamic Load balancing Parallel Lookup (CLPL), which is referred to as CLPL in this paper. By using logical caches, the traffic can be allocated to each TCAM relatively evenly. Unfortunately, CLPL doesn't cover routing compression and update, and its lookup mechanism can also be improved to some extent.

With regard to **iii)**, current solutions mostly either only focuses on i), or only on ii), but seldom emphasize update, maybe because the update was not so frequent before. However, according to our data mining results, the update issue is becoming more and more serious: the received updates messages of the backbone routers have reached 35K per second in the traffic peak, which will potentially make the traditional algorithm not applicable.

The system performance will be optimized, only if the three issues are solved simultaneously. In order to achieve this goal, there are still several aspects which need to be improved.

1) The routing table should be compressed to reduce hardware cost.

2) The amount of redundant prefixes should be further reduced.

3) The frequent interactions between data plane and control plane should be avoided.

4) The update algorithm should be studied profoundly.

Towards the above four targets, we propose a complete set of solutions -- CLUE. CLUE consists of a routing table compression algorithm (which leverages to our previous work, ONRTC), an improved parallel lookup mechanism based on CLPL, and a new incremental update mechanism, which involves the trie, TCAM and redundant prefixes update. The design philosophy of CLUE is that we should not view the three issues isolatedly and statically, so as to avoid one-sidedness. In other words, only an organic combination of the three aspects can make the forwarding plane of routers work well.

The first part of CLUE, i.e., the previously proposed ONRTC algorithm [2], compresses the routing table size to 70% of its original size. More importantly, it benefits the second and the third part greatly, for prefix overlap is eliminated.

The second part of CLUE is an improvement of CLPL. After compressed by ONRTC, the routing table is not overlapped any more, and thus a lot of advantages show up: 1) the domino effect in the TCAM update process will never happen again; 2) the priority encoder is no longer needed -- this not only reduces hardware cost, but also reduces the latency of TCAM lookup; 3) TCAM partitions can be split exactly evenly without redundancy; 4) The compression has saved a part of TCAM's hardware resource. In addition, we make the following improvements: 1) Dynamic Redundancy (DRed) *i* doesn't cache TCAM *i*'s prefixes, for TCAM *i* and DRed *i* will never be looked-up simultaneously, and thus 1/4 TCAM space can be saved when using four TCAMs; 2) because of Longest Prefix Match (LPM), when DRed is missed, the destination IP address must be sent to control plane to compute the prefix which should be stored in DRed by using RRC-ME algorithm [11]. Because overlap is eliminated by ONRTC, RRC-ME algorithm is no longer needed in CLUE. We just need to put the prefixes which are hit in the TCAM into DRed. Therefore, the interactions between control plane and date plane caused by DRed update can be totally avoided.

The third part of the CLUE is a brand new whole update algorithm, including trie update, TCAM update, and DRed update. In order to evaluate the update performance accurately and objectively, TTF (Time to Fresh) is defined here. TTF means the average computing time to update a message, which includes TTF1 (TTF-trie): update time of the trie, TTF2 (TTF-TCAM): update time of TCAM, and TTF3 (TTF-DRed): update time of redundant prefixes. TTF indicates a router's sensitivity to the changes of the network state. The smaller the TTF is, the more sensitive the router will be.

To summarize, the primary contributions of this paper lie in the following aspects:

- We propose an integrated problem – CLUE and a solution set with the same name. CLUE solves the above three issues simultaneously, improving the system performance.
- We present a complete mathematical proof to bound the speedup factor of CLUE, and verify the mathematical conclusion by experimental results.

The remaining parts of the paper are organized as follows. Section II surveys the related work. An improved mechanism based on CLPL is elaborated in Section III. Section IV illustrates the fast incremental update algorithm of the whole system. Extensive evaluation on CLUE over a large-sized real trace is conducted in Section V. Finally we conclude this paper in Section VI.

II. RELATED WORK

As mentioned above, three major issues are involved: routing table compression, routing lookup, and update.

A. Compression Algorithm

With respect to compression algorithm, many researches have been conducted to compress routing table, and the representative papers are [5-8], which only focus on routing table compression. These approaches will meet the following difficulties when being applied to TCAM lookup, given they don't solve the prefix overlap:

1) Layout in TCAM: Prefixes must be stored in TCAM ordered by their length, and a priority encoder is required to choose the longest one.

2) Update Handling: When update messages arrive, many prefixes will probably be moved to idle space to hold the new inserted ones, we call this domino effect. Some researches manage to resist domino effect, but redundant prefixes are unavoidably involved.

3) Power Consumption: one major shortcoming of TCAM is its high power consumption. An effective solution

is TCAM partition. There are mainly two partition algorithms: ID-bit partition [9, 10] and range partition [14], but [14] does not give the details to determine the partitions. Therefore, Dong Lin et al. proposed sub-tree partition [3] to implement range partition. ID-bit partition algorithm cannot split the table evenly. Sub-tree partition can work better, but it will introduce redundant prefixes.

If prefix overlap is eliminated, the above problems can be handled gracefully: 1) prefixes can be arbitrarily stored in TCAM; 2) the priority encoder is not needed any longer; 3) domino effect will never happen; 4) TCAM can be split strictly evenly without any redundancy.

There are mainly two approaches to reduce prefix overlap [12, 13]. In [12], the routing table is divided into two parts: the overlapping part and the non-overlapping part, which can only reduce overlap. To the best of our knowledge, only leaf-pushing algorithm proposed in [13] can totally eliminate overlap, but it will substantially incur the expansion of routing table.

Therefore, we have proposed ONRTC algorithm to construct optimal non-overlap routing tables, which is detailed in [2].

B. Routing Lookup Algorithm

With respect to routing lookup, R. Panigrahy et al. in [14] proposed a parallel searching method by employing multiple TCAM chips without any load balancing mechanisms, this approach cannot fully increase the speedup factor. That explains when eight TCAM chips were employed for parallel lookup operations, but only a speedup factor of five was achieved, given the lookup requests were not evenly distributed. In pursuit of an ultra-high lookup throughput, Kai Zheng et al. proposed a load balancing mechanism [10] based on 'pre-selected' redundant prefixes. The authors assume that the lookup traffic distribution over IP prefixes can be derived from the traffic traces statistically in a long period. Then a greedy algorithm with 25% more TCAM entries is proposed to optimally balance the traffic among the four TCAMs. The problem lies: 1) different network segments exhibit different traffic patterns, it's hard to come up with a universal law; 2) statistics in the past does not predict the future well. Thus in the worst case, when the traffic is bursty, the average throughput may decrease a lot.

Based on the data analysis collected from [15], Dong Lin et al. observed that the average overall bandwidth utilization was low but the Internet traffic could be very bursty. In order to handle the worst case, CLPL uses logical caches (which actually mean Dynamic Redundancy (DRed)) in place of the static redundant prefixes to achieve dynamic load balancing.

C. Incremental Update Mechanism

Speaking of incremental update, there are three aspects: i) trie update; ii) TCAM update; ii) DRed update, if DRed is used.

As for trie update, because SLPL and CLPL adopt no compression algorithm, thus their trie update is fast; while ONRTC algorithm is adopted in this paper, and the detailed incremental update algorithm is elaborated in [2].

As for TCAM update, one big obstacle is the domino effect, and many researchers strive to reduce it. In [16], the routing table is split into partitions according to the next hop. Each partition holds a collection of all the prefixes which share the same next hop, thus there is no need to keep the prefixes sorted in one partition. In this way, domino effect can be reduced. However, more than one prefix will be matched, thus the priority encoder is still needed. In addition, it cannot achieve power efficiency. In [17], all prefixes are split and allocated into different single-match TCAMs based on the ancestor-descendant relationship among them. It presents an algorithm to guarantee that each single-match TCAM generates at most one match for a given destination IP address. However, when a new prefix is inserted, each single-match TCAM may be required to move some of its existing prefixes to another TCAM in order to maintain a disjoint set. What's worse, power efficiency cannot be achieved.

As for DRed update, CLPL adopts LRU policy; as for routing update, SLPL and CLPL didn't mention it. Therefore, Bin Zhang et al. in [12] changed CLPL in the following two aspects:

1) A TCAM-chip is used as a DRed in place of CLPL's four logical caches. It is evident that this change will degrade the whole system's performance. This can be confirmed again by their experimental results in Figure 8 in [12]. The experiments use 12 TCAM chips, and half of the experimental results show that the speedup factor is less than 11, while 11 is the result of CLPL's worst case.

2) Bin Zhang et al. argued that SLPL and CLPL neglected an important factor—the update mechanism. Thus Bin Zhang et al. put the overlapping part of prefixes on the top of the TCAM and at the bottom of the next TCAM, while putting the non-overlapping part in the middle of the TCAM. This approach can reduce the domino effect to a certain extent. In contrast, in our paper we eliminate domino effect totally.

Therefore, we propose CLUE to handle these issues. As far as we know, this is the first effort on a full-dimensional system optimization for the three major routing table problems.

III. PARALLEL ROUTING LOOKUP MECHANISM

With regard to parallel lookup, the naive idea is to duplicate the routing table multiple copies while using round-robin lookup. This is obvious of expensive hardware cost and high power consumption. A smart idea [3, 10] is to divide the routing table into small partitions (by using partition algorithm), which are then allocated into TCAMs. A corresponding load balancing mechanism should be implemented, for the traffic load of each TCAM is not evenly distributed among TCAMS. The differences of different parallel lookup mechanisms lie in the partition algorithm, lookup mechanism, and redundancy mechanism for load balancing, which are detailed below.

A. Partition Algorithm

CLPL's sub-tree partition [3] algorithm outperforms SLPL's ID-bit partition algorithm [10]. Unfortunately, they both introduce redundancy. In our research, because prefix

overlap is eliminated by ONRTC, partition algorithm of CLUE becomes very simple, and the redundancy is no longer needed. The new partition algorithm of CLUE consists of two steps:

Step I: compute the partition size. Suppose the size of routing table is *M* and the partition count is *n*, then the size of each partition is *M/n*.

Step II: traverse the trie by inorder, and put every M/n prefixes to each TCAM partition.

It can be concluded that our new partition algorithm is much more simple and faster than ID-bit partition algorithm and sub-tree partition algorithm while guaranteeing high memory utilization.

B. *Parallel Lookup Mechanism*

The detailed implementation architecture of the parallel lookup engine is presented in Figure 1 (it appears originally in Figure 9 in [3]), which is an improved architecture based on CLPL mechanism. We here further make improvements and the new mechanism working process is described below (each step is marked by the number in Figure 1):

Step I, when an IP packet arrives, its destination IP address is sent to the Indexing Logic.

Step II, the Indexing Logic returns a partition number which tells the 'home TCAM' containing the matching prefix.

Step III, a tag (the sequence number) is attached to the IP address, since this mechanism may cause disorder.

Step IV, the IP address with the tag and partition number is delivered into the Adaptive Load Balancing Logic.

Step V, this is the most important step of our mechanism -- Dynamic Redundancy for load balancing. To be specific, each TCAM is split into partitions, and one partition is used as dynamic redundancy (DRed), as shown in Figure 1. The work mechanism of Dynamic Redundancy for load balancing is as follows:

a) If the queue of its home TCAM is not full, the incoming IP addresses will be looked-up in its home TCAM;

b) If the queue of its home TCAM is full, then the incoming IP addresses will be sent to the idlest queue. One important point is worth being mentioned: this kind of IP addresses will be only looked-up in the corresponding DRed, NOT the home TCAM. No IP address will be looked-up both in home TCAM and the corresponding DRed. So in CLUE, DRed *i* doesn't store TCAM *i*'s prefix. By this way, CLUE needs smaller redundancy, but has the same hit rate with CLPL.

c) If the incoming IP address is missed in DRed, it will be sent back and repeat the above a).

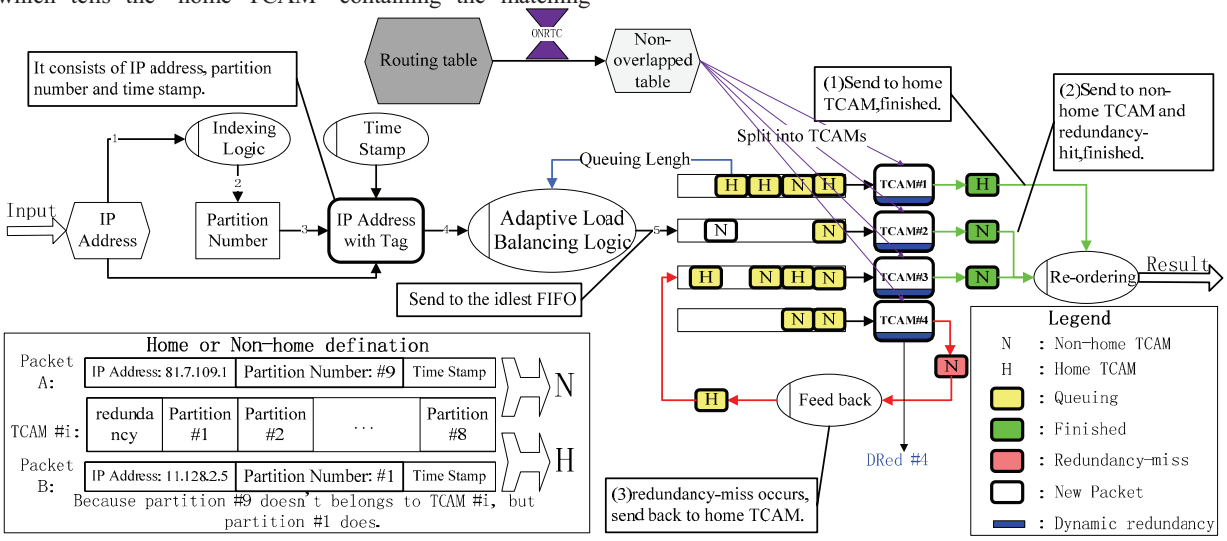


Figure 1. Improved parallel lookup mechanism

C. *Novel Dynamic Redundancy Mechanism*

To achieve load balancing, SLPL adopts statistical redundancy, while CLPL adopts logical caches. For CLPL, as mentioned above, no IP address will be looked-up in both home TCAM and the corresponding dynamic redundancy (DRed). As a result, this is NOT really a cache, and we use the word 'DRed' in place of CLPL's logical caches, but the DRed is updated by the cache mechanism. According to the following analysis and experimental results in Section V, DRed outperforms logical caches a lot.

Generally speaking, cache mechanisms can be divided into two categories: caching destination addresses (IPs) [18-20] and caching prefixes [21]. The results in these papers have demonstrated that caching prefixes is more efficient, and this is also in accord with our experimental results. Therefore, we adopt caching prefixes, as CLPL did.

However, one big obstacle of caching prefix is prefix overlap. Because of LPM, if an inner node is matched, the corresponding prefix cannot be sent to DRed.

For example, as shown in Figure 2, a prefix = 100000 is looked-up in the home TCAM, and the LPM result returns p (p=1*). However, p cannot be sent to DRed, because its child node q owns a different next hop. It is obvious that p'=100*

should be sent to DRed. This approach is called RRC-ME algorithm [21], which is adopted in CLPL.

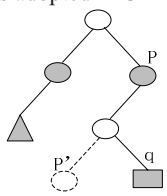


Figure 2. RRC-ME algorithm

For RRC-ME algorithm, one important issue is worth being mentioned here. Although RRC-ME algorithm is simple, its update algorithm is not an easy task. The update algorithm mentioned here means that when the routing table updates, the DRed must update as well. This process must visit SRAM several times, which incurs additional overhead.

In addition, CLPL adopts RRC-ME algorithm, which causes frequent interactions between data plane and control plane, while CLUE doesn't, and the details are as follows.

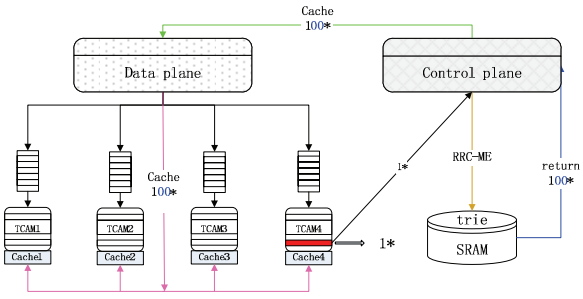


Figure 3. The DRed update process of CLPL's mechanism

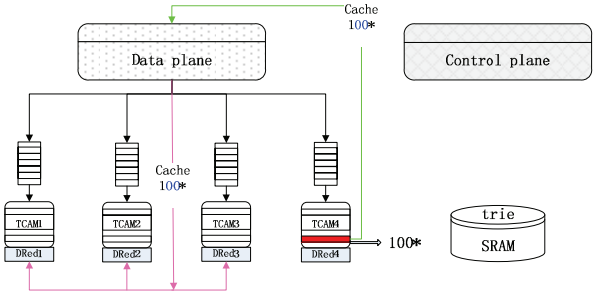


Figure 4. The DRed update process of CLUE's mechanism

The DRed update process of CLPL is shown in Figure 3. If an incoming IP is looked-up in TCAM1, then the LPM prefix is 1*. Prefix 1* must be sent to control plane, which executes RRC-ME algorithm by traversing the trie stored in SRAM, and returns 100*. Then 100* is sent to data plane, and is inserted into the four logical caches. It can be noted that the DRed update process will execute RRC-ME algorithm, and frequent interactions happen, disturbing routing lookup a lot.

As aforementioned, DRed *i* and TCAM *i* will never be looked-up simultaneously, thereby CLUE reduces CLPL's DRed size by the rule that DRed *i* doesn't cache TCAM *i*'s prefixes, but the hit rate doesn't decline. With regard to the four TCAM chips, the DRed size of CLUE is reduced into 3/4 of that of CLPL.

Because overlap is eliminated, RRC-ME algorithm is no longer involved. Our new DRed process is shown in Figure 4. If an incoming IP address is looked-up in TCAM1, the result of LPM is 100*, then 100* is sent to data plane directly, and is inserted into the other three DReds. As a result, control plane will not be involved in the process of DRed update process. In this way, the update process of DRed is faster and more efficient.

To sum up, our new DRed Mechanism can achieve smaller DRed size and avoid frequent interactions between data plane and control plane.

D. Lower Bound of the System Performance

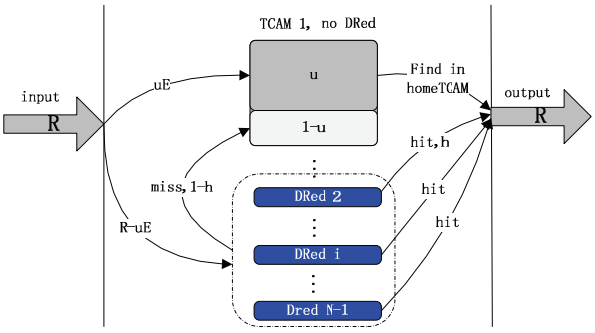


Figure 5. Parallel lookup in the worst case

With regard to the lower bound of the system performance, Dong Lin et al. has presented a formal mathematical proof, which is flawed. Therefore, a complete formal mathematical proof is given here, and the proof conclusion is in accord with the corresponding experimental results.

Our deduction is under the following two premises:

1) The update cost is ignored.

2) The TCAM1 is always working.

The two premises are practically feasible. Regarding premise 1), Dong Lin et al. show that when the cache size is set to 1024 and only one cache-missed element is updated within 5000 clock cycles, the system can still easily achieve 100% throughput [3]. In addition, each routing update only causes one shift in TCAM, that is, *O(1)* time complexity by CLUE. Therefore, the time cost of routing update is negligible. Premise 2) holds true via a policy that keeps the queue of TCAM1 never empty.

In the worst case of this parallel system, all the traffic is delivered to a single TCAM, which is TCAM1 in Figure 5. It suggests that $TCAM_2$~$TCAM_{N-1}$ is idle, only $DRed_2$~$DRed_{N-1}$ are still working. The definitions of the symbols are described below:

N means the number of TCAM chips used in this system;

R means the maximum input traffic;

E means the processing ability of each TCAM;

u means the percentage of TCAM1's processing ability used to handle the traffic which goes directly to TCAM1;

1-u means the percentage of processing ability used to handle the packets missed in DRed;

h means the hit rate of the DRed;

t means the speedup factor of this system.

Firstly, it is easy to get each symbol's range:

$$\begin{cases} 0 < u < 1 \\ 0 < h < 1 \\ 2 \le N \end{cases}$$

Secondly, when all TCAMs work at their best, the system can handle the maximum workload R.

$$\left.\begin{matrix} R = tE \\ uE + (N-1)E = R \end{matrix}\right\} \Rightarrow t = N + u - 1 \quad (1)$$

$$\Rightarrow t \ge N - 1 \quad (2)$$

Thirdly, TCAM1 preserves 1-u processing ability to handle DRed-missed traffic.

$$\left.\begin{matrix} R = tE \\ (R - uE) * (1 - h) = (1 - u)E \end{matrix}\right\} \Rightarrow h = \frac{t-1}{t-u} = \frac{t-1}{N-1}$$

$$h = \frac{t-1}{t-u} = \frac{t-1}{N-1} = \frac{N-2+u}{N-1} \quad (3)$$

$$\Rightarrow h \ge \frac{N-2}{N-1} \quad (4)$$

According to (1) and (3), we get that

$$t = (N-1)h + 1 \quad (5)$$

Then

$$t \ge N - 1$$

As long as

$$h \ge \frac{N-2}{N-1}$$

According to [18-21] and our experiment results, the hit rate of (N-2)/(N-1) can be easily achieved.

According to (5), it can be concluded that in the worst case

$$h \propto t$$

It suggests that in real traffic, $t \ge (N-1)h + 1$ always holds true. This conclusion is in consistent with subsequent experimental results (see Figure 16).

IV. The New Incremental Update Mechanism

When an update message arrives, the routing table should be updated as fast as possible. Specifically, the whole update process can be divided into three steps (see Figure 6): 1) trie update; 2) TCAM update; 3) DRed update. When the three steps are all finished, the update message takes into effect. Then how to evaluate the performance of incremental update? Time to Fresh (TTF) is defined in this paper, including TTF1 (TTF-trie), TTF2 (TTF-TCAM), and TTF3 (TTF-DRed).

The update involved in TCAM will interrupt routing lookup, resulting in decrease of lookup rate. Therefore, TTF2 and TTF3 are more important than TTF1.

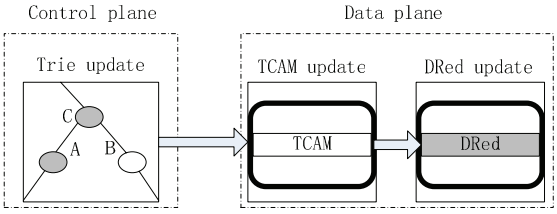


Figure 6. The whole incremental update process of CLUE.

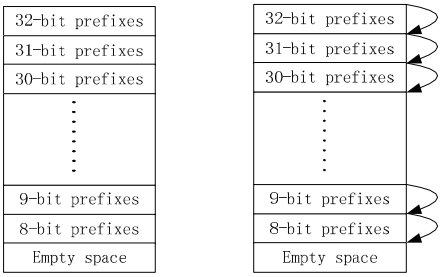
A. Trie Update

The performance of trie update is evaluated by TTF1 (TTF-trie), which means the average computing time of updating the trie. If no compression algorithm is adopted, TTF1 is minimal, and is regarded as ground-truth.

Because CLPL mechanism adopted no compression algorithm, thus its TTF1 is minimal, and TTF1-CLPL is defined to represent its TTF-trie. To be different, we adopt previously proposed ONRTC algorithm to compress the trie, and the update time is represented by TTF1-CLUE. The process of incremental update always keeps the trie non-overlap. Experimental results show that TTF1-CLUE is a little bit longer than TTF1-CLPL.

B. TCAM Update

Generally speaking, in order to reduce domino effect, the common method is to keep some redundancy at the end of TCAM. A naive solution [22] is shown in Figure 7(a). When a prefix is inserted, it will move all the following prefixes one by one, thus has a time complexity *O(n)* in the worst case, which is clearly undesirable.



(a) a naive solution (b) a classical solution

Figure 7. Two approaches to reduce domino effect

A classical solution [22] is based on the observation that two prefixes with the same length can be placed interchangeably. This suggests that the domino effect can be reduced. As shown in Figure 7(b), there is only a partial ordering constraint among all prefixes. This approach enables an empty memory location to be found in at most 32 prefix shifts. This is a classical method without extra overhead, thus we assume it is adopted in CLPL, and is selected to be compared with CLUE. Experimental results show that this solution needs 14.994 shifts in average.

The previous work [3, 10] didn't emphasize incremental update too much. After overlap is eliminated, the updating method of TCAM becomes clear and simple: when inserting

a prefix, just write it to the end of TCAM; when deleting a prefix, just cut the last prefix to replace it. Therefore, CLUE needs one shift at most to handle an update message.

The performance of TCAM update is evaluated by TTF2 (TTF-TCAM), which indicates the update time of TCAM. Whatever partition algorithms are adopted, TCAM update must wait till the trie update is finished. The current partition algorithms probably need to change more than one prefix when one update message arrives.

It is obvious that our new TCAM update mechanism is much more efficient than all the traditional TCAM update mechanisms. This conclusion is consistent with the subsequent experimental results.

C. DRed Update

After the update of TCAM, in order to guarantee synchronization and correctness, the DRed must be updated, too. As mentioned above, CLPL adopts RRC-ME algorithm and its update algorithm.

When inserting or deleting a prefix in home TCAM, RRC-ME algorithm must look up the trie and find all the prefixes which may be changed by this update. During the process, SRAM must be visited several times, which is a waste of time.

In contrast, when inserting a prefix in home TCAM, CLUE's DRed needs no change; when deleting a prefix, CLUE just lookups it in the DRed. If it exists, just delete it; otherwise, do nothing.

It is obvious that CLUE is much more efficient than CLPL in DRed update. This conclusion is also in accord with the subsequent experimental results. TTF is the sum of TTF1, TTF2, and TTF3. The whole TTF comparison between CLPL and CLUE is given in the subsequent experimental results.

V. Experimental Result

A. Experimental Settings

1) Trace

TABLE I. Locations of Routers.

ID	Location	ID	Location
rrc01	LINX, London	rrc11	New York (NY), USA
rrc03	AMS-IX, Amsterdam	rrc12	Frankfurt, Germany
rrc04	CIXP, Geneva	rrc13	Moscow, Russia
rrc05	VIX, Vienna	rrc14	Palo Alto, USA
rrc06	Otemachi, Japan	rrc15	Sao Paulo, Brazil
rrc07	Stockholm, Sweden	rrc16	Miami, USA

The RIB packets are taken from www.ripe.net [1] at RIPE NCC, Amsterdam, which collects default free routing updates from peers. In order to objectively test the performance of ONRTC algorithms, the RIB packets at 8:00 on October 1 in 2011 from 12 routers are selected. (There are 16 routing tables available in www.ripe.net, but four of them don't update to present). Table I shows the routers' location.

In the routing update experiment, two traces are selected. In order to measure TTF-ratio, the update data from 2011.10.01/08:00 to 2011.10.02/08:00 is selected.

With regard to real traffic, trace from [23] is selected. The traffic from 20:59 to 21:14 on 2011.02.17 in Chicago is downloaded and parsed.

Our lab has a router prototype with TCAM (CYNSE70256) in its linecards. CYNSE70256 supports 256K entries with 36-bit width. It can operate at a speed of up to 41.5 MHz by looking up 36 bit-width entries. It suggests that each lookup costs:

$$1s/41.5\text{MHz} \approx 24\text{ns}$$

Generally speaking, the update time of each lookup is roughly equal to the time of moving a prefix. Therefore, 24ns is regarded as the time cost of moving one prefix in TCAM.

2) Computer Configuration

Our experiments are carried out on a windows XP sp3 machine with Pentium (R) Dual-Core CPU 5500@2.80GHz and 4G Memory.

B. Experiments on Compression by ONRTC

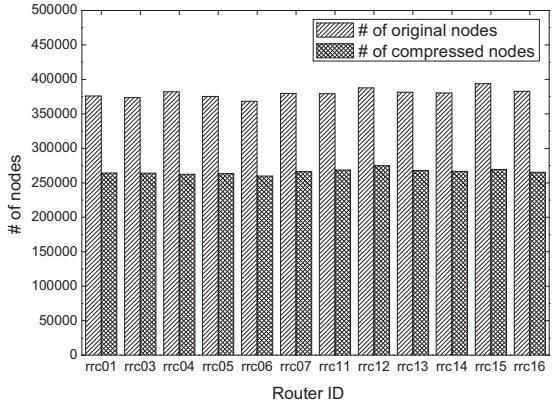


Figure 8. FIB size before and after compression on 12 routers.

The compression results of ONRTC of 12 routers are shown in Figure 8. The taller bars are the original FIB size, while the lower bars are the FIB size after compression by ONRTC algorithm. According to the results, the compressed prefix number is 71% of the original in average, and the compression time is around 39 milliseconds.

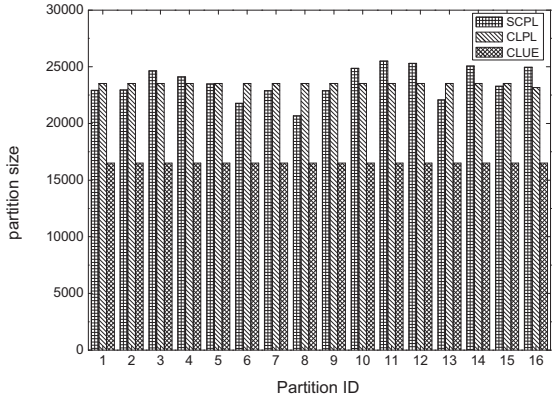


Figure 9. partition comparison among the three algorithms.

Figure 9 shows the partition results of three algorithms: SCPL algorithm, CLPL algorithm, and CLUE algorithm. The same experiments are conducted on 12 routers, and only one is shown in the figure, given the results are similar. As shown in the figure, SCPL cannot split prefixes evenly, and CLPL split prefixes evenly at the cost of redundancy. In contrast, CLUE splits prefixes evenly with no redundancy, with much fewer prefixes in one partition than both SCPL and CLPL. Besides, as the number of partitions rises, SCPL and CLPL introduces more redundancy (see Figure 6 in [3]), while CLUE still has no redundancy.

C. Experiments on TTF

The x-axis of Figure 10~14 stands for the arrival time of update messages. For example, '201010231945' means 2010.10.10/23:19:45.

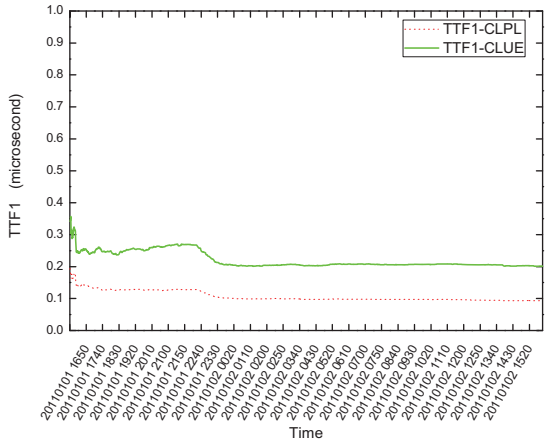


Figure 10. TTF1 comparison between CLPL and CLUE.

Figure 10 shows TTF1 (TTF-trie) of CLUE (ONRTC) and CLPL (ground-truth). It can be observed that TTF1 of CLUE is a little longer than ground-truth. TTF1 of CLUE ranges from 0.1924 microseconds to 0.3574 microseconds with a mean of 0.2210 microseconds. Because TTF1 doesn't interrupt routing lookup, a litter bigger TTF1 of CLUE doesn't influence the system performance.

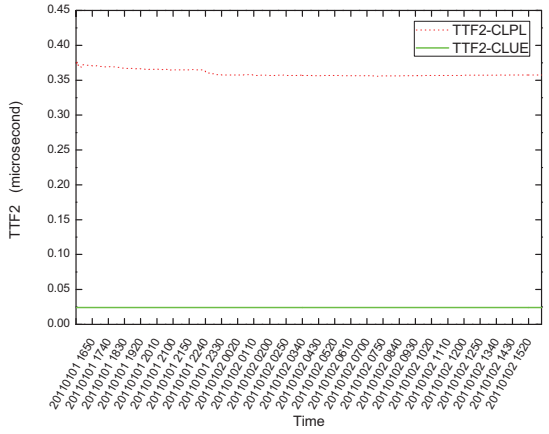


Figure 11. TTF2 comparison between CLPL and CLUE.

Figure 11 shows TTF2 (TTF-TCAM) of CLUE and the general method (see Figure 7(b)). As mentioned in experimental settings, 24ns is regarded as the time cost of moving one prefix in TCAM. TTF2 of CLPL ranges from 0.3558 microseconds to 0.3782 microseconds with a mean of 0.3598 microseconds. In contrast, as mentioned above, CLUE needs only one shift (*O(1)*) to handle an update message, which means 0.024 microseconds for each update.

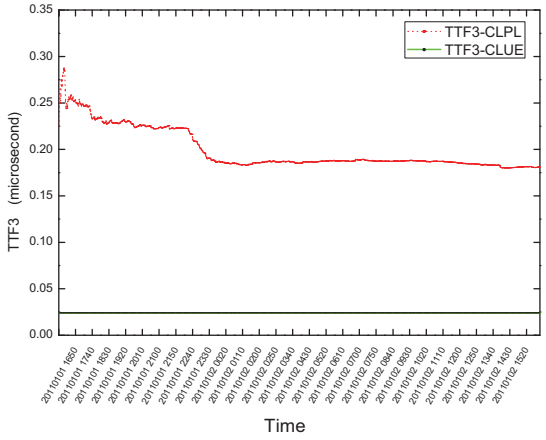


Figure 12. TTF3 comparison between CLPL and CLUE.

To evaluate the TTF3, we plot TTF-DRed in Figure 12. TTF3 of CLUE still maintains 0.024 microseconds; while TTF3 of CLPL ranges from 0.1802 microseconds to 0.2878 microseconds with a mean of 0.1993 microseconds. In other words, TTF3 of CLPL is 8.3 times of that of CLUE in average, and 11.99 times in the worst case.

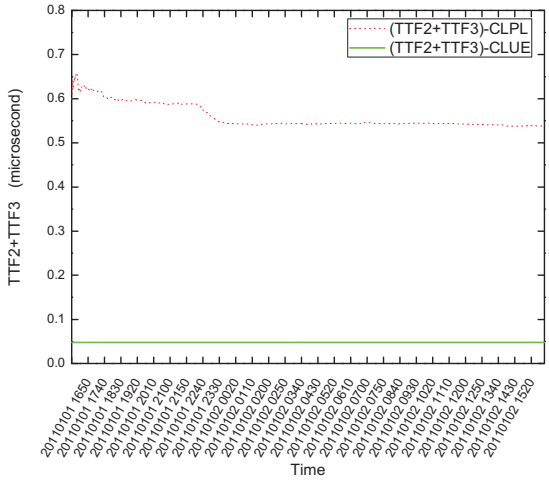


Figure 13. TTF2+TTF3 comparison between CLPL and CLUE.

As aforementioned, TTF2 and TTF3 are more important than TTF1, because TTF1 is the time cost in the control plane which doesn't interrupt routing lookup. In other words, TTF2 and TTF3 influence the system performance much more than TTF1. Therefore, the comparison of TTF2+TTF3 between CLPL and CLUE is shown in Figure 13. Results

show that TTF2+TTF3 of CLUE is 4.29% of CLPL in average and 3.65% in the worst case.

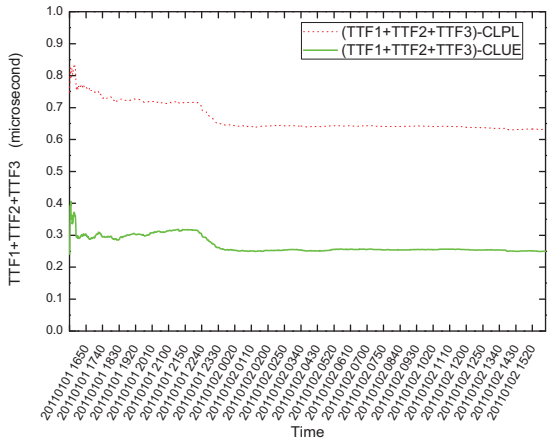


Figure 14. TTF1+TTF2+TT3 comparison between CLPL and CLUE

TTF, which is the sum of TTF1, TTF2, and TTF3, measures a router's sensitivity to the changes of the network state. Figure 14 shows the TTF of CLPL and CLUE. In the figure, the TTF of CLPL ranges from 0.6303 microseconds to 0.8342 microseconds with a mean of 0.6664 microseconds. In contrast, TTF of CLUE is only 0.2690 microseconds in average. In other words, TTF of CLPL is 234% of that of CLUE.

D. Experiments on Parallel Lookup

TABLE II. WORKLOAD ON DIFFERENT PARTITIONS AND TCAM CHIPS.

# of TCAM chips	#of Bucket	Range Low	Range High	Percent of partition	Percent of TCAM
1	2	38.103.176.0	61.91.89.255	21.92%	77.88%
	12	97.69.128.0	119.46.79.255	10.57%	
	20	194.133.118.0	196.11.124.255	9.18%	
	…				
2	23	202.30.78.0	203.128.191.255	4.52%	17.43%
	31	216.207.89.0	255.255.255.255	3.32%	
	8	72.9.88.0	77.79.211.255	3.13%	
	…				
3	16	168.87.144.0	183.87.78.255	0.81%	4.54%
	13	119.46.80.0	134.75.216.255	0.72%	
	5	65.68.16.0	66.133.181.255	0.70%	
	…				
4	11	91.209.9.0	97.69.127.255	0.08%	0.16%
	10	85.95.88.0	91.209.8.255	0.07%	
	0	0.0.0.0	12.177.231.255	0.00%	
	…				

As shown in TABLE II, routing table from rrc01 is split into 32 partitions evenly by CLUE. After test by real traffic, the workload of each partition is shown in Column 'Percent of partition'. It can be observed that workload among different partitions varies a lot. To simulate bursty traffic, the partitions are sorted by the workload percentage in descending order. The first 8 partitions are mapped to $TCAM_1$, while the second, third and the fourth 8 partitions are mapped to TCAM 2, 3, 4, respectively. This is a possible mapping situation with extremely uneven workload among TCAMs.

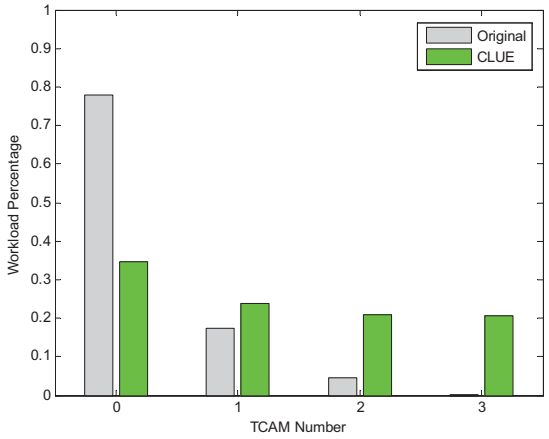


Figure 15. Load balancing of workload distribution by CLUE.

The grey bars labeled 'Original' in Figure 15 show the extremely uneven workload distribution of Table II. An experiment using this distribution is designed to evaluate the function of CLUE's load balancing. In the simulation process, each TCAM takes 4 clocks to process a packet, while a packet arrives per clock. The FIFO is set to 256 and redundancy size is set to 1024 prefixes. The green bars show the traffic distribution balanced by CLUE. It can be seen that the green bars labeled 'CLUE' are much more even than 'Original'. It can be concluded that CLUE can achieve excellent load balancing performance even in the worst case.

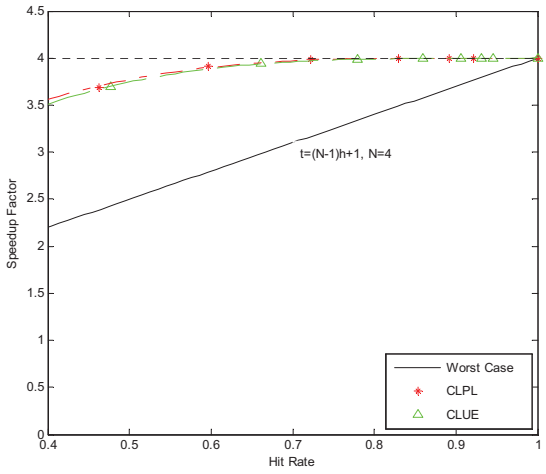


Figure 16. Speedup factor comparison between CLPL and CLUE and the worst case

Figure 16 shows the relationship between Hit Rate and Speedup Factor. It is a comparison among CLPL, CLUE,

and the worst case in theory. The dotted lines of CLPL and CLUE are the results of cubic curve fitting. Both CLPL and CLUE are much better than the worst case, which is consistent with previous theory results. This figure suggests that the speedup factor rises as hit rate rises. In terms of CLPL and CLUE, the same Speedup Factor will be achieved by the same hit rate, because they almost overlap.

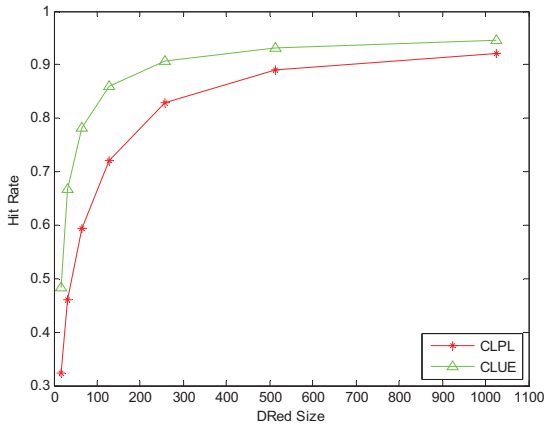


Figure 17. Hit rate comparison between CLPL and CLUE.

The relationship between DRed Size and Hit Rate is plotted in Figure 17. The top curve is the result of CLUE, and the other one belongs to CLPL. It indicates that CLUE achieves much higher Hit Rate than CLPL with the same DRed Size. Whereas Figure 16 shows Hit Rate determines Speedup Factor, then it can be indirectly concluded that CLUE achieves much higher Speedup Factor than CLPL with the same DRed Size.

VI. CONCLUSIONS

Due to the explosive increase of Internet volume and traffic, routing tables in backbone routers have been increasing approximately 15% in size annually [4]. Meanwhile, the link transmission speed of backbone routers has increased to tens of gigabit-per-second. Consequently, the backbone routers are facing CLUE: routing table *C*ompression, fast routing *L*ookup, and fast incremental *U*pdat*E*.

Because traditional algorithms seldom cover the three problems simultaneously, we propose a complete set of solutions -- CLUE. The design philosophy of CLUE is that we should not view the three problems isolatedly and statically, avoiding one-sidedness. Firstly, CLUE adopts ONRTC algorithm, which supports parallel routing lookup and fast incremental update. Secondly, several improvements are made based on CLPL mechanism, achieving lower hardware cost. Thirdly, a novel whole update algorithm TTF is defined and evaluated, including TTF-trie, TTF-TCAM, and TTF-DRed. Extensive experimental results show that CLUE needs much less hardware resource and shorter update time to achieve the same speedup factor compared with CLPL.

REFERENCES

[1] RIPE Network Coordination Centre. http://www.ripe.net/data-tools/stats/ris/ris-raw-data.

[2] Tong Yang, Ting Zhang, Shenjiang Zhang and Bin Liu. Constructing Optimal Non-overlap Routing Tables. In Proc. ICC, 2012.

[3] Lin, D., Zhang, Y., Hu, C., Liu, B., Zhang, X., Pao, D. Route Table Partitioning and Load Balancing for Parallel Searching with TCAMs. In Proc. IPDPS, 2007.

[4] X. Meng, Z. Xu, B. Zhang, G. Huston, S. Lu, and L. Zhang. IPv4 Address Allocation and the BGP Routing Table. ACM SIGCOMM Computer Communication Review, vol. 35, pp. 71–80, January 2005.

[5] R. Draves, C. King, S. Venkatachary, and B. D. Zill. Constructing Optimal IP Routing Tables. In Proc. IEEE INFOCOM, 1999.

[6] X. Zhao, Y. Liu, L. Wang, and B. Zhang. On the Aggregatability of Router Forwarding Tables. In Proc. IEEE INFOCOM, 2010.

[7] Heeyeol Yu. A memory- and time-efficient on-chip TCAM minimizer for IP lookup. DATE '10 Proceedings of the Conference on Design, Automation and Test in Europe, 2010.

[8] Qing Li, Dan Wangy, Mingwei Xu, Jiahai Yang. On the Scalability of Router Forwarding Tables. Nexthop-Selectable FIB Aggregation. In Proc. IEEE INFOCOM, 2011.

[9] F. Zane, G. Narlikar, A. Basu. CoolCAMs: Power-Efficient TCAMs for Forwarding Engines. In Proc. INFOCOM, 2003.

[10] Zheng, K., Hu, C., Lu, H., Liu, B. A TCAM-based distributed parallel IP lookup scheme and performance analysis. IEEE/ACM Trans. Netw. 14, 863–875, 2006.

[11] Mohammad J. Akhbarizadeh and Mehrdad Nourani. Efficient Prefix Cache for Network Processors, High Performance Interconnects 2004, pp.41-46, August 2004.

[12] Bin Zhang, Jiahai Yang, Jianping Wu, Qi Li, Donghong Qin. An Efficient Parallel TCAM Scheme for the Forwarding Engine of the Next-generation Router. In Proc. IFIP/IEEE IM, 2011.

[13] V. Srinivasan and G. Varghese. Fast IP lookups using controlled prefix expansion, ACM TOCS, vol. 17, pp. 1–40, Feb. 1999.

[14] R. Panigrahy, S. Sharma. Reducing TCAM Power Consumption and Increasing Throughput. Proceedings of HotI 2002, pp.107-112, August 2002.

[15] Abilene. http://www.abilene.iu.edu/noc.html.

[16] E. Ng and G. Lee. Eliminating sorting in ip lookup devices using partitioned table. In The 16th IEEE International Conf. on Application Specific Systems, Architecture and Processors, 2005.

[17] K. Jinsoo and K. Junghwan. An efficient ip lookup architecture with fast update using single-match tcams. In WWIC, 2008.

[18] Woei-Luen Shyu, Cheng-Shong Wu, and Ting-Chao Hou. Efficiency Analyses on Routing Cache Replacement Algorithms, ICC'2002, Vol.4, pp.2[24]2-2236, April/May 2002.

[19] Tzi-cker Chiueh,Prashant Pradhan. High-Performance IP Routing table Lookup Using CPU Caching. In Proc. INFOCOM, 2003.

[20] Bryan Talbot, Timothy Sherwood, Bill Lin. IP Caching for Terabit Speed Routers. In Proc. GLOBECOM, 1999.

[21] Mohammad J, Akhbarizadeh and Mehrdad Nourani. Efficient Prefix Cache for Network Processors. High Performance Interconnects 2004, pp.41-46, August 2004.

[22] D. Shah and P. Gupta, Fast incremental updates on ternary-CAMs for routing lookups and packet classification. In Proc. Hot Interconnects 8, Aug. 2000, pp. 145–153.

[23] The CAIDA Anonymized 2011 Internet Traces - <20110217> Colby Walsworth, Emile Aben, kc claffy, Dan Andersen, http://www.caida.org/data/passive/passive_2009_dataset.xml