A Time Efficient Algorithm Based on Bloom Filters for Longest Prefix Matching in IP Lookups

Ming Yu
School of Information and Communication Engineering, Dalian University of Technology, Dalian, China
Email: yu_ming1111@dlut.edu.cn

Dongju Wang
School of Information and Communication Engineering, Dalian University of Technology, Dalian, China
Email: juflower123@126.com

***Abstract*—A time efficient algorithm based on Bloom filters is proposed to solve the problem of longest prefix matching in IP lookups. It is distinguished from the existing algorithms by three features. Firstly, a first-byte indexing table is established before querying the Bloom filters, so that the impact of positive false inherent to Bloom filters on IP lookups is reduced. Secondly, only twelve Bloom filters are required for prefix query, as is based on the uniform distribution of various prefix lengths. Thirdly, update of routing tables is supported by associating a counter with each bit in the bit vectors of the Bloom filters. Performance analysis and experimental results show the efficacy of the proposed algorithm in decreasing the time complexity of IP lookups.**

***Index Terms*—Bloom filters, IP lookups, longest prefix matching, time efficient algorithm**

I. Introduction

Due to the explosive growth of the Internet, Internet core routers need to forward packets as fast as possible to keep up with the ever-increasing optical transmission rates. This requires more efficient implementations of packet processing functions. At the same time, CIDR (Classless Inter-Domain Routing) was widely adopted to prolong the life of IPv4 (Internet Protocol version 4) [1]. CIDR requires Internet routers to search variable-length address prefixes in order to find the longest matching prefix of the IP destination address and retrieve the corresponding forwarding information for each packet traversing the router. These operations are commonly referred as IP lookup, which is regarded as a bottleneck in improving the performance of Internet core routers.

Longest prefix matching techniques have drawn extensive attention from IT researchers all over the world in the past ten years. This is due to the fundamental role it plays in the performance of IP lookups. Generally, longest prefix matching operations need to determine the next hop information for incoming packets. It is determined by first performing a longest prefix match of the destination IP address of incoming packets against a set of prefixes stored in a prefix table. If a match is found, the next hop information associated with the matched prefix is retrieved. For IPv4 addresses, the prefix table can typically have a few hundred thousand prefixes. The lengths of these prefixes vary from 8 to 32 bits. For IPv6 addresses, the prefix lengths vary from 16 to 64 bits.

Nowadays, designers of commercial routers tend to use TCAM (Ternary Content Addressable Memory) devices for IP lookups in order to keep pace with optical link speeds despite their larger size, cost, and power consumption relative to SRAM (Static Random Access Memory) devices [2,3]. Generally, the performance bottleneck in longest prefix matching algorithms employing RAM (Random Access Memory) is the number of dependent memory accesses required to complete the IP lookups. Dependent memory accesses must be performed sequentially, whereas independent memory accesses may be performed in parallel. Therefore, some algorithms allow dependent memory accesses to be masked via pipelining, with each stage accessing an independent memory bank or port. However, this quickly becomes an expensive option.

In this paper, we focus on improving the time efficiency of IP lookups. Our study is based on Bloom filters which are typically used for efficient exact match searches. The main contribution of this paper is to propose a time efficient algorithm based on Bloom filters to solve the problem of longest prefix matching in IP lookups. Different from the existing algorithms based on Bloom filters for IP lookups, the proposed algorithm supports update of the route tables by adopting the counting Bloom filter. Moreover, a table of {FirstByte, PrefixLength} is used to reduce the number of Bloom filters without increasing the size of hash tables. As a result, the time efficiency of IP lookups is improved by the proposed algorithm.

Rest of this paper is organized as follows. Section II discusses the related works. Section III discusses the principles of IP lookups based on Bloom filters. Section IV presents a time efficient algorithm based on Bloom filters for longest prefix matching in IP lookups. Section

doi:10.4304/jcp.8.10.2724-2729

V describes the experiments on the proposed algorithm and analyzes the experimental results. Section VI concludes this paper.

II. Related Works

IP lookup is a well-researched topic because of its essential role in Internet core routers. Nowadays, challenges in implementing IP lookup operations are in accommodating large table sizes, achieving the 150 million or more lookups per second needed for the new 100Gbps interfaces while keeping memory, power consumption and board footprint low [4,5]. Although TCAM devices are employed in most high-performance routers to perform lookups at optical link speeds, their high power dissipation and large footprint are major disadvantages. Moreover, it has long been argued that the brute-force way of searching prefixes in TCAM leads to very inefficient storage and uses too many bits per prefix.

On the contrary, algorithmic approaches for IP lookups can achieve high rates with compact storage needs. Compact storage permits the use of fast memories such as SRAMs economically and also the effective use of on-chip memory to achieve high throughputs.

Currently, the existing IP lookup algorithms can be classified into 4 groups. Respectively, they are multi-branch trie-based algorithms, binary search on address prefix length, prefix range search and TCAM lookup. The key in designing multi-branch trie-based algorithms is to find a compromise scheme between lookup speed and storage space [6]. This is due to a contradiction for such algorithms in time complexity and space complexity [7]. Algorithms based on binary search on address prefix length are time efficient, but update of these algorithms is difficult [8,9]. Algorithms based on prefix range search have good scalability to various prefix lengths, but they do not support update of routing tables [10]. TCAM lookup can implement fast routing table lookups, but it brings significant disadvantages in terms of cost and power consumption [11,12].

In recent years, applications of Bloom filters in IP lookups have attracted wide attention. In the process of IP lookups, algorithms based on Bloom filters can predict the prefix lengths of the IP addresses to be looked up, and accomplish one memory access for one lookup. Dharmapurikar *et.al*. proposed to store the prefixes with different lengths in different Bloom filters [7]. The IP address to be processed is put into all Bloom filters in parallel and then the hash tables with different prefix lengths are looked up to determine the next hop information of the IP address. It is pointed out in [7] that both the number of Bloom filters and the time to search the hash tables have great influence on the performance of the IP lookups by algorithms based on Bloom filters.

Lijun adopts a method of prefix expanding to reduce the number of Bloom filters used in IP lookups. By this method, the number of hash table access is reduced effectively [13]. However, prefix expanding increases the size of hash tables, which increases the time to search hash tables.

Song *et.al*. [4] adopt a type of improved Bloom filter in their method and implement it in hardware. Yet, this method has a complex structure, and different lengths of IP prefixes need different hash functions to complete Bloom filter storage. Moreover it is difficult to find so many independent hash functions.

We draw inspiration from the related works that reduce the number of Bloom filters or the time to search the hash tables may be a possible way to improve IP lookups based on Bloom filters. Our algorithm to be discussed in this paper is designed based on this understanding.

III. Principles of IP Lookups Based on Bloom Filters

The Bloom filter is a data structure used for representing a set of elements succinctly. A filter is first "programmed" with each element in a set, and then queried to determine the membership of a particular element [2]. It was formulated by Burton H. Bloom in 1970 [14], and is widely used for different purposes such as web caching, intrusion detection, and content based routing [15,16]. For the convenience of the readers, we explain the theory behind Bloom filters in this section.

A. Basic Bloom Filters

A basic Bloom filter includes a m-bits vector $\boldsymbol{V}$ and a group of hash functions [17,18]. For a set of n elements, for example $\boldsymbol{X}=\{x_1, x_2, \ldots, x_n\}$, the typical operations of a basic Bloom filter are inserting the elements to the Bloom filter and querying the Bloom filter for element membership. k hash functions, $h_1(x)$, $h_2(x)$, …, $h_k(x)$, are used to complete both operations. Fig.1 illustrates the working flow of the basic Bloom filter and further explanation of it is described as follows.

1) Inserting an element to a basic Bloom filter [19].
   (i) Initialization the Bloom filter: set $\boldsymbol{V}$ to zero.
   (ii) For $\forall x_i \in X$, compute the hash values of x_i by $h_1(x_i)$, $h_2(x_i)$, ⋯, $h_k(x_i)$.
   (iii) Set the corresponding bits in $\boldsymbol{V}$ to "1", that is, $\boldsymbol{V}[h_1(x_i)]=\boldsymbol{V}[h_2(x_i)]=\cdots=\boldsymbol{V}[h_k(x_i)]=1$.
2) Element membership query
   (i) For an element y, compute the hash values of it by $h_1(y)$, $h_2(y)$, ⋯, $h_k(y)$.

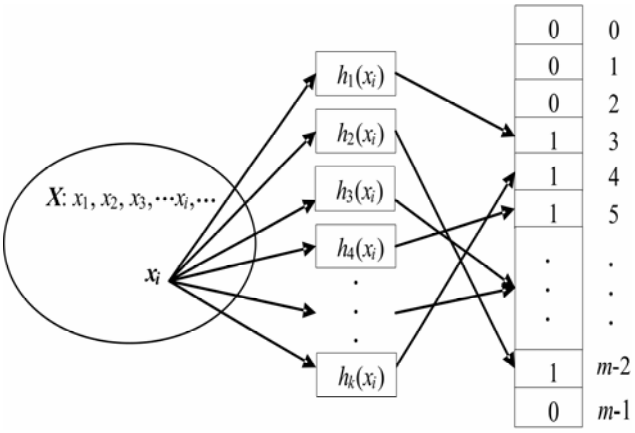

Figure 1 Working flow of the basic Bloom filter.

(ii) Check the corresponding bits in array $\boldsymbol{V}$. Rules for confirmation of element membership of y in $\boldsymbol{X}$ are as follows.
If $\boldsymbol{V}[h_1(y)]\&\boldsymbol{V}[h_2(y)]\&\cdots\&\boldsymbol{V}[h_k(y)]=1$, the membership of y in $\boldsymbol{X}$ is confirmed.
Else, y is considered being not included in $\boldsymbol{X}$.

In an element membership query, a positive false may happen when the membership of y in $\boldsymbol{X}$ is confirmed while it doesn't belong to $\boldsymbol{X}$ [20].

B. Counting Bloom Filters

For a basic Bloom filter, deleting a particular entry requires the corresponding k hashed bits in the bit vector $\boldsymbol{V}$ be set to zero. This could disturb other elements which are hashed to any of these bits. Therefore, it is impossible to delete an entry stored in a basic Bloom filter. In order to solve this problem, the idea of the counting Bloom filters was proposed by L. Fan *et.al*. [21]. Counting Bloom filters provide a way to implement a "delete" operation for a basic Bloom filter without recreating the filter afresh. A counting Bloom filter maintains a vector of counters corresponding to each bit in the bit vector $\boldsymbol{V}$. Whenever an element is added to or deleted from the filter, the counters corresponding to the k hash values are incremented or decremented, respectively. Especially, the corresponding bit in the bit vector $\boldsymbol{V}$ is set to "1" when a counter changes from zero to one. Similarly, the corresponding bit in the bit-vector is set to zero when a counter changes from one to zero.

C. IP Lookups Based on Bloom Filters

The key operation for IP lookup algorithms based on Bloom filters is to store the routing entries in different hash tables according to the prefix lengths of the destination IP addresses. If a packet needs forwarding, its destination IP address is firstly sent to the Bloom filters in a parallel mode. Then, a matching vector is got. Based on this matching vector, the IP lookup algorithm probes the corresponding hash tables, and obtains the next hop information. Fig.2 gives the general steps of IP lookup algorithms based on Bloom filters.

Ⅳ. The Proposed Algorithm

As we have mentioned in Section III, positive false may disturb the element membership query when Bloom filters are used to determine whether an element belongs to a specific set. Therefore, it is a key problem in designing an IP lookup algorithm based on Bloom filters to decrease the number of positive false. We hope to solve this problem by decreasing the number of Bloom filters. Taking this into consideration, we propose to add a first-byte indexing table before querying the Bloom filter. This table is indexed by the first bytes of all the IP addresses in the route table, and the data structure of its entries is designed as {FirstByte, PrefixLength 1, PrefixLength 2, ⋯, PrefixLength n}. Besides, we propose to associate a counter with each bit in the bit vector of each basic Bloom filter so that update of the routing table is supported in our proposed algorithm. This inspiration comes from the counting Bloom filters.

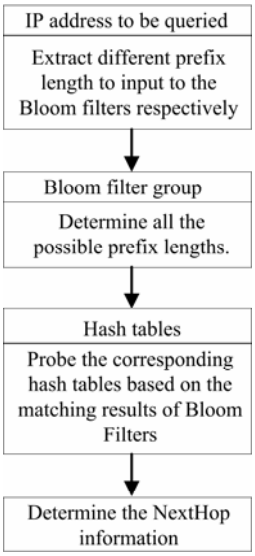

Figure 2 General steps of IP lookup algorithms based on Bloom filters.

A. Searching Operations

As we know, the IPv4 address is 32 bits long. However, the distribution of different prefix lengths is not uniform. This conclusion can be supported by the data supplied in http://bgp.potaroo.net/as2.0/bgp-active.html. We illustrate these data in Fig.3. As we can see from this figure, IP addresses with the prefix length of less than 8 bits or greater than 30 bits do not exist or take a very small percentage of the total routing table entries. This forms the basis of our algorithm to reduce the number of Bloom filters.

In our proposed algorithm, we employ 12 Bloom filters and adopt two 12-bits vector (denoted by $\boldsymbol{V}_1$ and V_2) to indicate respectively the querying results on the first-byte indexing table and the matching results on each of the 12 Bloom filters. For example, the IP addresses with the prefix length of 8~14 bits are stored in a Bloom filter which is represented by the first bit in $\boldsymbol{V}_1$ (V_2); the IP addresses with the prefix length of 25~30 bits are stored in a Bloom filter which is represented by the last bit in $\boldsymbol{V}_1$ (V_2); and IP addresses with the prefix length of 15~24 bits are stored respectively in 10 Bloom filters which are represented by the rest 10 bits in $\boldsymbol{V}_1$ (V_2). Corresponding to each Bloom filter, there is a hash table which contains the next hop information of various prefixes and the data structure of which is designed as {Prefix, NextHop}.

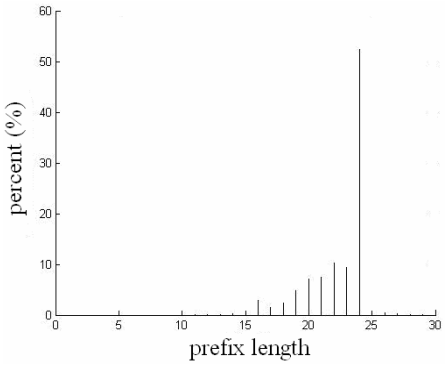

Figure 3 Distribution of prefix lengths.

The searching operations are carried out as follows. Firstly, the first byte of the input IP address is extracted. Then, the extracted byte is searched in the first-byte indexing table. If no matching items are found in this table, the IP packet is sent to the default port. Otherwise, all possible prefix lengths corresponding to the first byte of the IP address are extracted and the corresponding bits of $\boldsymbol{V}_1$ are set to "1". Then, the IP address to be queried is sent to those Bloom filters of which the corresponding bits in $\boldsymbol{V}_1$ are set to "1". After querying the Bloom filters, the matching result forms the bit vector $\boldsymbol{V}_2$ where a bit is set as "1" if the prefix of the IP address is matched by the corresponding Bloom filter. If $\boldsymbol{V}_2$=**0**, the IP packet is sent to the default port. Otherwise, the hash tables are probed based on $\boldsymbol{V}_2$ according to the longest matching principle. If the probe is successful, the next hop information is obtained and the IP packet is sent to the corresponding port. Otherwise, the IP packet is sent to the default port. Fig.4 gives the working flow of the searching operations.

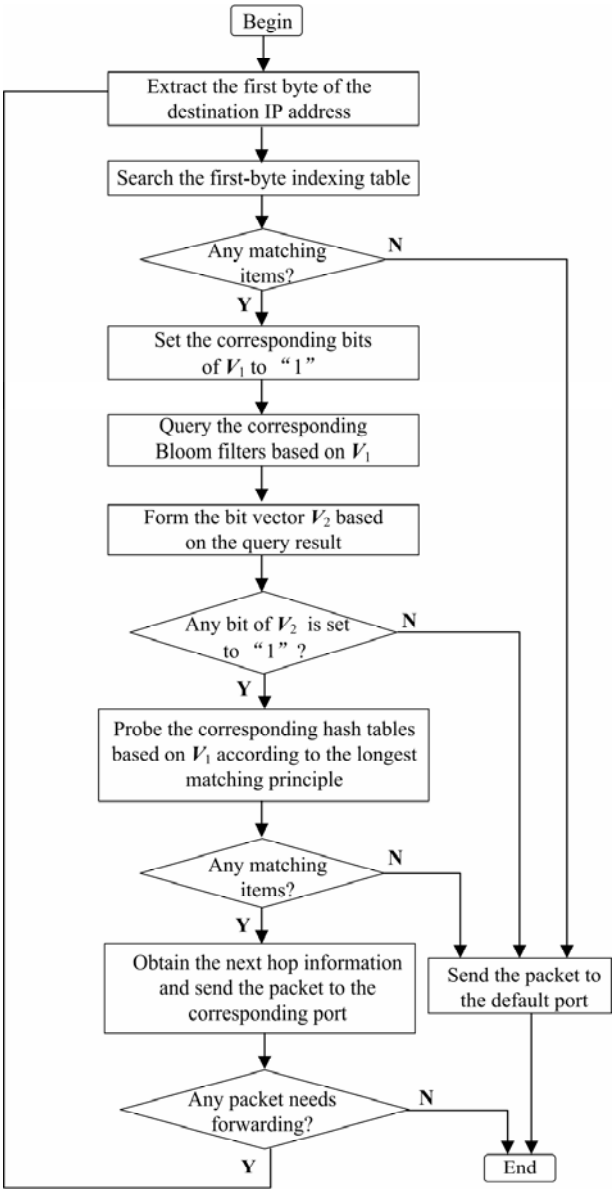

Figure 4 Working flow of the searching operations.

B. Update Operations

Based on the assumption that update of the routing table is infrequent, IP lookup algorithms based on Bloom filters seldom support adding or deleting entries from the routing tables. In the proposed algorithm, update of routing tables is supported by associating a counter with each bit in the bit vectors of the Bloom filters. Since the space occupied by the counter is large and the update operations are relatively infrequent, counters of the Bloom filters are put in the off-chip memory and controlled by independent controllers.

The operation of adding a new prefix is done as follows. Firstly, the proposed algorithm will add it to the basic Bloom filter according to its length. Then, the associated counters will be increased by 1. If the value of the counters is one, the corresponding bit in the bit vector is set to 1. If the counter is already nonzero before the addition, nothing will be done to the bit vector of the Bloom filter. Finally, the proposed algorithm will add the right {Prefix, NextHop} entry to the corresponding hash table, and update the first-byte indexing table at the same time.

The operation of deleting a prefix is contrary to the adding operation. Firstly, the proposed algorithm will delete the {Prefix, NextHop} entry from the corresponding hash table, and update the first-byte indexing table at the same time. Then, the associated counters will be decreased by 1. If the value of the corresponding counter is zero, the corresponding bit of its associated bit vector is set to zero. If the associated counter is nonzero, nothing will be done to the bit vector of the Bloom filter.

C. Performance Analysis

Usually, two performance indexes are used to compare performance of different IP lookups algorithms based on Bloom filters. Respectively, they are time complexity and space complexity. Due to the development of computer technologies, influence of space complexity on IP lookup algorithms is decreasing [22]. Researchers pay more attention on improving the performance of IP lookup algorithms by reducing their time complexity. For the proposed algorithm in this paper, there are two factors that influence the time complexity of the algorithm. Respectively, they are the number of times to probe the hash tables and the size of the hash tables. Here, we take an IPv4 address lookup as an example to analyze the time complexity of the proposed algorithm.

For the proposed algorithm, the number of times to probe the hash tables is determined by two vectors, that is, V_1 and V_2. Assume: (i) the number of Bloom filters corresponding to V_1 is B; (ii) all Bloom filters share the same positive false probability f; (iii) the number of Bloom filters for the prefixes longer than l in the vector V_2 is B_l. Then, the average number of times to probe the hash tables when matching a prefix of length l is

$$E_l = B_l * f \tag{1}$$

In the worst case, the value of E_l which is represented by $E_{l\text{-max}}$ can be computed as

$$E_{l\text{-max}} = B*f \quad (2)$$

Here, $E_{l\text{-max}}$ is used to estimate the maximum number of times to probe the hash tables for each IP lookup. Since the computation of $E_{l\text{-max}}$ takes the positive false into consideration, the upper bound of the average number of times to probe the hash table for each IP lookup of all the input IP addresses can be estimated as

$$E_{\text{avg}} \leq E_{l\text{-max}} + 1 = B*f + 1 \quad (3)$$

In expression (3), the number "1" stands for the probe at the matching prefix length. Besides, it is possible that an IP address creates false matches in all the Bloom filters. In this case, the number of times to probe the hash tables can be estimated as

$$E_{\text{worst}} = B \quad (4)$$

Let: (i) f_i be the positive false probability of the i^{th} Bloom filter; (ii) k_i be the number of hash functions in the i^{th} Bloom filter; (iii) m_i be the size of the i^{th} Bloom filter; (iv) n_i be the number of prefixes stored in the i^{th} Bloom filter. According to Rothenberg *et.al.*[19], k_i can be computed as follows:

$$k_i = (m_i/n_i)\text{In}2 \quad (5)$$

Let: (i) M be the embedded memory available for the Bloom filters; (ii) N be the number of prefixes supported by the IP lookup system. Then, the positive false probability of the i^{th} Bloom filter can be estimated as

$$f_i = (\frac{1}{2})^{(\frac{m_i}{n_i})\ln 2} = (\frac{1}{2})^{(\frac{M}{N})\ln 2}, \quad \forall i \in (1,2,3,\cdots,31) \quad (6)$$

The size of hash tables is also a factor that affects the time complexity of the proposed algorithm. Let the size of hash table be N_{H} and the time of each hash probe be t. Then, t is in direct proportion to N_{H}, which means the smaller the hash table is, the lower the time complexity of the proposed algorithm is.

V. Experimental Results and Analysis

Firstly, experiments based on Matlab are carried out to illustrate results of the performance analysis on the proposed algorithm. Figure 4 gives the experimental results.

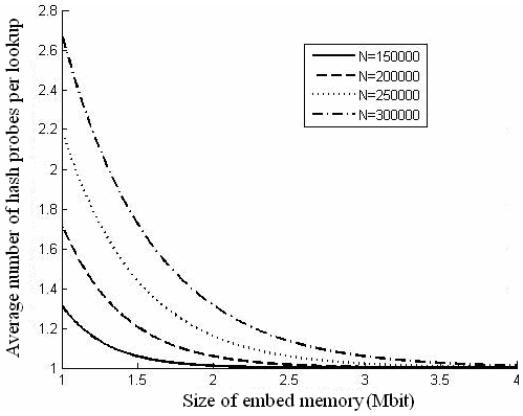

Figure 4 Illustration of the performance analysis on the proposed algorithm

As we can see,

1) The average number of times to probe the hash tables for each IP lookup is greater than one, which is consistent with the expression (3).

2) The average number of times to probe the hash tables for each IP lookup decreases with the increasing of size of the embedded memory used by the routing system.

3) The average number of times to probe the hash tables for each IP lookup is close to 1 when the embedded memory is increased to 4M bits. This means the positive false probability f is close to zero.

Secondly, several real route tables from China Unicom, China Telecom and China Mobile are used to validate the efficiency of the proposed algorithm. The total number of entries in these routing tables is more than 250,000. Size of the embedded memory for Bloom filters is configured to be 4Mbits. The number of hash functions for each Bloom filter is configured to be 6. The proposed algorithm is implemented based on the Eclipse integrated development environment and Java. Table I compares the experimental results and the theoretical results estimated by the expression (3) and Equation (6) on the averaged number of times to probe the hash tables for each IP lookup.

Finally, we show the relationship between the size of hash tables and the time needed to complete a hash table probe by averaging the results of ten experiments of which 100000 hash table probes were carried out. The results are shown in Table II. As we can see from it, the larger the hash table is, the longer the time needed to complete a hash table probe is. This is consistent with the analysis result described in Section IV.

TABLE I
Experimental Results and Theoretical Results on the Averaged Number of Times to Probe the Hash Tables for Each IP Lookup

The number of entries in the hash table	Experimental results	Theoretical results
198,754	1.0011	1.0002
200,164	1.0007	1.0003
217,342	1.0013	1.0006
254,526	1.0034	1.0022
263,445	1.0047	1.0029

TABLE II
Experimental Results on the Relationship between the Size of Hash Tables and the Time Needed to Complete a Hash Table Probe

Size of hash table	Time per hash table detection(*ns*)
2524	75.43
3279	83.21
3812	94.314
4462	101.75
4983	119.38

VI. CONCLUSIONS

In this paper, a time efficient algorithm based on Bloom filters is proposed to solve the problem of longest prefix matching in IP lookups. Different from the existing algorithms based on Bloom filters for IP lookups, the proposed algorithm set a first-byte indexing table before querying the Bloom filter, so that the number of Bloom filters is reduced without increasing the size of hash tables. Moreover, the proposed algorithm supports update of the route tables by associating a counter with each bit in the bit vectors of the Bloom filters. As a result, the time efficiency of IP lookups by the proposed algorithm is improved. In the future, we plan to make further improvement and optimization on the proposed algorithm and employ it in IP lookups for IPv6 addresses.

ACKNOWLEDGMENT

This work was supported by (1) National Natural Science Foundation of China (Grant No.61172059); (2) the Scientific Research Foundation for Ph.Ds of Liaoning Province, China (Grant No.20111022).

REFERENCES

[1] S. Fuller, T. Li, J. Yu, and K. Varadhan, "Classless inter-domain routing (CIDR): an address assignment and aggregation strategy", RFC 1519, September 1993

[2] Sarang Dharmapurikar, Praveen Krishnamurthy, David E. Taylor, "Longest prefix matching using bloom filters", In proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '03), pp.201-212.

[3] Syed Iftekhar Ali, M.S. Islam, "An energy efficient design of high-speed ternary CAM using match-line segmentation and resistive feedback in sense amplifier", Journal of Computers (Finland), vol.7, no. 3, pp.567-577, 2012.

[4] Haoyu Song and Fang Hao and Murali S. Kodialam and T. V. Lakshman, "IPv6 Lookups using Distributed and Load Balanced Bloom Filters for 100Gbps Core Router Line Cards", In Proceedings of IEEE INFOCOM 2009, pp. 2518-2526.

[5] Niu Yun, Liji WU, Xiangmin ZHANG, "An ipsec accelerator design for a 10Gbps in-line security network processor", Journal of Computers (Finland), vol.8, no.2, pp.319-325, 2013.

[6] Hoang Le, Weirong Jiang, Prasanna V.K., "Memory-Efficient IPv4/v6 Lookup on FPGAs Using Distance-Bounded Path Compression", In proceeding of 2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp.242-249, 2011.

[7] Dharmapurikar,S., Krishnamurthy,P., Taylor,D.E. "Longest Prefix Matching Using Bloom Filters", Computer Communication Review, ACM SIGCOMM, vol.33, no.4, pp.201-212, 2003.

[8] Marko Zec, Luigi Rizzo, Miljenko Mikuc, "DXR: towards a billion routing lookups per second in software", Computer Communication Review, ACM SIGCOMM, vol.42, no.5, pp.29-36, 2012.

[9] Xia ZHUGE, Koji NAKANO, "Direct Binary Search Based Algorithms for Image Hiding", International Journal of Digital Content Technology and its Applications, AICIT, vol. 6, no. 23, pp.457-466, 2012.

[10] Yu-Chen Kuo, Chih-Cheng Li, "A dynamic indirect IP lookup based on prefix relationships", In proceeding of 2012 18th Asia-Pacific Conference on Communications (APCC), pp.674-679, 2012.

[11] Hui Yu, Jing Chen, Jianping Wang, Zheng, S.Q., Nourani, M., "An improved TCAM-based IP lookup engine", In proceeding of 2008 International Conference on High Performance Switching and Routing (HSPR 2008), pp.1-5, 2008.

[12] A.Bremler-Barr, D.Hendler, "Space-Efficient TCAM-Based Classification Using Gray Coding", IEEE Transactions on Computers, IEEE, vol.61, no.1, pp.18-30, 2012.

[13] Jun Li, Bloom Filter Improvement and Its Applications in Distribute System, NanKai University. China, 2007.

[14] B. H. Bloom. "Space/Time Trade-offs in Hash Coding with Allowable Errors", Communications of the ACM, ACM, vol.13, no.7, pp.422-426, 1970.

[15] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey", Internet Mathematics, Taylor & Francis, vol.1, no.4, pp.485-509, 2004.

[16] Jiangqing Wang, Rongbo Zhu, "Efficient intelligent optimized algorithm for dynamic vehicle routing problem", Journal of Software (Finland), vol.6, no.11, pp. 2201-2208, 2011.

[17] Chakrit Wannachakrit, Toni Anwar, "Managing Network System by Bloom Filter", International Journal of Digital Content Technology and its Applications, AICIT, vol.6, no.22, pp.33-45, 2012.

[18] Ken Christensen, Allen Roginsky, Miguel Jimeno, "A new analysis of the false positive rate of a Bloom filter", Information Processing Letters, ELSEVIER, vol 110, no.21, pp.944-949, 2010.

[19] Rothenberg C.E., Macapuna C.A.B., Verdi F.L., Magalhães M.F., "The deletable Bloom filter: a new member of the Bloom family", IEEE Communications Letters, IEEE, vol.14, no.6, pp.557-559, 2010.

[20] Deke Guo, Yunhao Liu, Xiangyang Li, Panlong Yang, "False Negative Problem of Counting Bloom Filter", IEEE Transactions on Knowledge and Data Engineering, IEEE, vol.22, no.5, pp.651-664, 2010.

[21] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol", IEEE/ACM Transactions on Networking, IEEE, vol.8, no.3, pp.281-293, 2000.

[22] Zhiwei Ni, Junfeng Guo, Li Wang, Yazhuo, Gao, "An efficient method for improving query efficiency in data warehouse", Journal of Software (Finland), vol.6, no.5, pp. 857-865, 2011.

Ming Yu received the BS degree in electronics engineering in 1998 from Shandong University, China. He received the MS degree and Ph.D degree in information and telecommunication system in 2004 and 2008 from Xidian University, China. He is currently an associate professor in Dalian University of Technology, China. He is also a member of IEEE Computer Society. So far, he has 15 papers published in international journals. His research interests include network security, cloud computing and DoS defense.

Dongju Wang received the BS degree in computer science and information engineering in 2009 from Qufu Normal University, China. She is now a postgraduate in information and telecommunication system in Dalian University of technology, China. Her research interests include computer networks and DoS defense.