IEIE Transactions on Smart Processing and Computing, vol. 4, no. 3, June 2015
http://dx.doi.org/10.5573/IEIESPC.2015.4.3.163

On Reducing False Positives of a Bloom Filter in Trie-Based Algorithms

Ju Hyoung Mun and Hyesook Lim

Dept. of Electronics Engineering, Ewha Womans University / Seoul, Korea {jhmun@ewhain.net, hlim@ewha.ac.kr}

* Corresponding Author: Hyesook Lim

Received March 26, 2015; Accepted April 17, 2015; Published June 30, 2015

* *Regular Paper*

* *Extended from a Conference: The abstract version of this paper were presented at the ACM ANCS in October 2014. This present paper has been accepted by the editorial board through the regular reviewing process that confirms the original contribution.*

***Abstract*:** Many IP address lookup approaches employ Bloom filters to obtain a high-speed search performance. Especially, it has been recently studied that the search performance of trie-based algorithms can be significantly improved by adding Bloom filters. In such algorithms, the number of trie accesses can be greatly reduced because Bloom filters can determine whether a node exists in a trie without actually accessing the trie. Bloom filters do not have false negatives but have false positives. False positives can lead to unnecessary trie accesses. The false positive rate must thus be reduced to enhance the performance of lookup algorithms applying Bloom filters. One important characteristic of trie-based algorithms is that all the ancestors of a node are also stored. The proposed algorithm utilizes this characteristic in reducing the false positive rate of a Bloom filter without increasing the size of the memory for the Bloom filter. When a Bloom filter produces a positive result for a node of a trie, we propose to check whether the ancestors of the node are also positives. Because Bloom filters have no false negatives, the negatives of any of the ancestors mean that the positive of the node is false. In other words, we propose to use more Bloom filter queries to reduce the false positive rate of a Bloom filter in trie-based algorithms. Simulation results show that querying one ancestor of a node can reduce the false positive rate by up to 67% with exactly the same architecture and the same memory requirement. The proposed approach can be applied to other trie-based algorithms employing Bloom filters.

***Keywords*:** IP lookup, Bloom filter, Binary search on levels

1. Introduction

Classless inter-domain routing (CIDR) architecture makes Internet protocol (IP) address lookup algorithms perform a task of finding the longest prefix matching for a given input IP address. Finding the longest matching prefix is much more complex than finding an exact match, and this should be performed at line-speed for every incoming packet [1]. As the Internet grows, the size of routing tables is also growing rapidly. It is an important challenge in designing Internet routers to develop efficient IP address lookup algorithms to work on large routing tables.

Bloom filters have been actively employed in various applications due to their compactness and simplicity [3]. Many IP address lookup algorithms employ Bloom filters [4, 5]. Especially adding Bloom filters to trie-based architectures has been recently proposed [5]. Reducing the false positive rate of Bloom filters is a good challenge to enhance the performance of these approaches.

We focus that every node in a trie is stored in trie-based algorithms [2]. This means that all the ancestor nodes of a node are stored in trie-based algorithms. By using this characteristic, we propose a novel algorithm that significantly reduces the false positives of a Bloom filter when checking for node existence without increasing the memory requirement of the Bloom filter. We verified the proposed approach for the architectures of the binary search on levels with a Bloom filter.

The rest of this paper is organized as follows. Section 2 briefly introduces related works. The proposed algorithm is explained in Section 3. The performance evaluation results using actual routing tables are shown in Section 4. This paper is concluded in Section 5.

2. Related Work

2.1 Bloom Filters

A Bloom filter provides a simple but highly space-efficient way of identifying the membership of a set. A Bloom filter is composed of an array of m bits, which contains the summary of members included in a set. For a given set $S=\{x_1,x_2,\cdots,x_n\}$, the programming procedure of a Bloom filter is as follows. First, every bit in the Bloom filter is initialized to zero. The k hash functions $h_i(x)$ for $1\le i\le k$ are required to program the Bloom filter. Hash indices obtained from these hash functions should be in the range of $0,\cdots,m-1$. In programming an element x_j for $1\le j\le n$ into a Bloom filter, the k bits indicated by k hash indices obtained for x_j are set to one. This programming procedure is repeated for every element included in set S.

The query procedure to check the membership of a given input also uses the same k hash functions. For an input y, the Bloom filter bits pointed by the indices acquired from k hash functions $h_i(y)$ for $1\le i\le k$ are checked. If all these bits are set, y is considered a member of set S. If any of these bits is zero, y is absolutely not a member of set S. Bloom filters do not have false negatives but have false positives. However, the false positive rate can be properly controlled by increasing the size of the Bloom filter and the number of hash functions accordingly. For n elements, the false positive rate p_f of an m-bit Bloom filter can be calculated as follows [3].

$$P_f=\left\{1-\left(1-\frac{1}{m}\right)^{kn}\right\}^k\approx\left(1-e^{-\frac{kn}{m}}\right)^k \quad (1)$$

The optimal number of hash functions for an m-bit Bloom filter of n elements can be calculated as follows.

$$k_{opt}=\frac{m}{n}\ln 2 \quad (2)$$

2.2 Binary Trie

A binary trie is a tree-based data structure which stores prefix information in a node of the trie [2]. The routing information of a prefix of length l is located in a node at level l of the trie, and the prefix value determines the path from the root node to the node. Fig. 1 shows an example of a trie of 7 prefixes: 000*, 010*, 1*, 1101*, 110101* 111*, and 11111*. Each prefix is allocated in the trie according to its value.

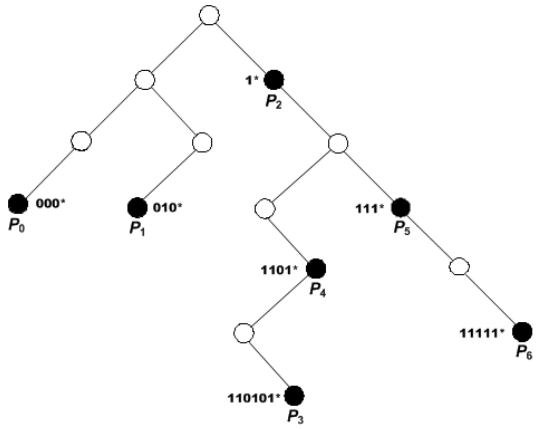

Fig. 1. A binary trie [2].

Searching the routing information corresponding to a given input is linearly performed by examining each address bit at a time along the trie, starting from the root node. Because IP address lookup problem is to find out the longest matching prefix, the search is finished when there is no edge to follow. For example, when an input IP address of 110111 comes, starting from the root node the search goes to the right node because the first bit of the input IP address is 1. The right node of the root is a prefix node, and hence the routing information of P_2 is remembered. The search continues to lower levels. Because the second bit of the input IP address is 1, the search goes to the right, and so on. At level 4, a matching prefix node, P_4, is remembered, and there is no branch to follow. Therefore the routing information for P_4 is returned. The number of memory accesses for an IP address lookup can be 32 for IPv4 in the worst-case scenario [1]. In order to reduce the number of memory accesses, adding a Bloom filter to trie-based architectures has been studied [5].

2.3 Algorithms Based on Binary Search on Trie Levels

To improve the search performance of the binary trie, binary search on trie levels has been introduced by Waldvogel et al. The Waldvogel's binary search on levels (WBSL) performs a binary search on levels of a trie. Each level of the trie is stored in a hash table. The existence of a node at each level leads a search to a longer level. When there is no matching prefix in the longer level, back-tracking should occur. To prevent back-tracking, a marker and the best matching prefix (BMP) information should be pre-computed in every internal node [6]. Fig. 2 shows the WBSL trie, and the order of binary search on levels is described on the right side of the figure. The figure also shows the pre-computed markers and BMPs. Levels 1, 3, 4, 5, and 6 have prefixes, and the middle of these levels is 3. Hence, level 3 is searched first. If a marker node is encountered, the BMP is remembered and the search goes to a longer level. Otherwise, if there is no matching node,

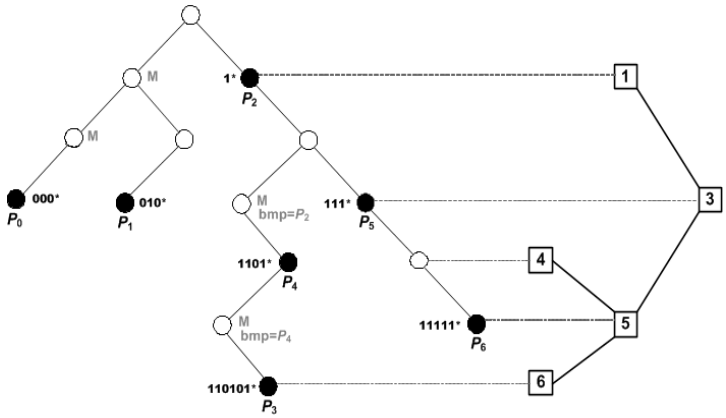

Fig. 2. Waldvogel's binary search on levels [2].

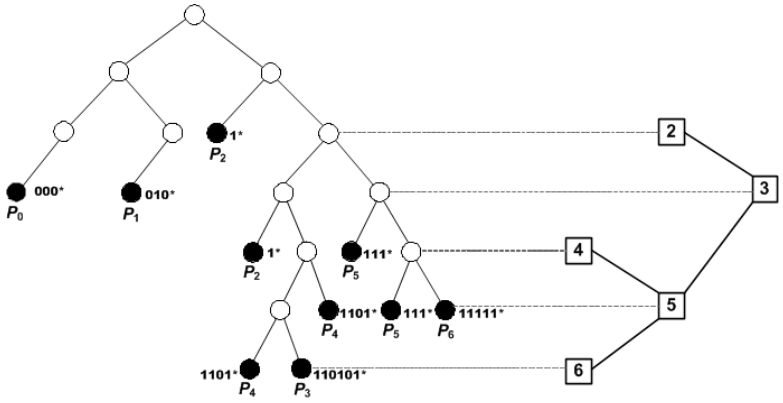

Fig. 3. Binary search on levels in a leaf-pushed trie (LBSL) [2].

the search proceeds to a shorter level. The search ends at the last level of access. For the same example of the input IP address 110111, the search encounters marker node 110 at level 3. Hence the search continues at level 5, and no matching node is found. Then the search goes to level 4 and encounters a prefix, P_4. There are no longer levels to be searched, so the search procedure is finished by returning the routing information of P_4.

Binary search on levels in a leaf-pushing trie (LBSL) has been proposed to remove the requirement for pre-computation of the markers and the best matching prefixes [7]. Leaf-pushing relocates every internal prefix to leaf nodes. Hence, an internal node guarantees that there is no prefix node at shorter levels. Therefore, the pre-computation is no longer required. Leaf-pushing removes the nesting relationship between prefixes. Hence, when a matching prefix node is found, the search can be finished immediately. Fig. 3 shows a trie which is the leaf-pushed version of the trie shown in Fig. 1. As shown in the figure, each internal node prefix is relocated to a leaf in the leaf-pushed trie. Therefore, the search procedure can be finished when a matching prefix is found. The order of binary search on levels is described in the right side of Fig. 3. In accessing a level, if a matching prefix is encountered, the search is immediately terminated. Otherwise, if a matching internal node is encountered, the search goes to a longer level. Otherwise, if there is no matching node, the search proceeds to a shorter level. For the same example of the input IP address 110111, the search encounters an internal node 110 at level 3. Hence, the search continues at level 5 and encounters a prefix, P_4. The search immediately terminates by returning the routing information for P_4.

To reduce the number of hash accesses, adding a Bloom filter to these BSL-based algorithms was proposed, and abbreviated as WBSL-BF and LBSL-BF [5]. In these algorithms, Bloom filters determine the existence of a node. Since Bloom filters do not have false negatives, the search can proceed to a shorter level without a trie access when the Bloom filter produces a negative for the query of a node.

3. The Proposed Algorithm

It is well known that the size of a Bloom filter and accordingly the number of hash indices should be increased in order to reduce the false positive rate. In this paper, we suggest a different way of reducing the false positive rate. Our proposed method is to use more queries by utilizing a characteristic of a trie: a node cannot exist without ancestor nodes. In performing the binary search on levels in a trie, every node at valid levels is stored. For the example trie in Fig. 1, every node at levels 1, 3, 4, 5, and 6

```
Search (dstIP) {
    min = minLevel; max = maxLevel;
    while (min ≤ max) {
        next = (min+max)/2;
        qBF0 = queryBF(dstIP, next);
        if (qBF0 is positive) {
            qBF1 = queryBF(dstIP, next-1);
            if (qBF1 is positive) {
                node = accessHash(dstIP, next);
                if (node == prefix node)
                    return node.info;
                else if (node == internal node)
                    min = next + 1;
                else /* false positive */
                    max = next - 1;
            }
            else /* when qBF1 is negative */
                max = next - 1;
        }
        else /* when qBF0 is negative */
            max = next - 1;
    }
}
```

Fig. 4. Search process of the proposed algorithm.

is stored. If a Bloom filter produces a positive result for a node at level 3, we propose to query level 1 in this example, which is an ancestor of the node at level 3, to check whether it is also a positive. Because Bloom filters have no false negatives, the negative of the ancestor means that the positive for the current level is false. Therefore, the false positive rate can be reduced. Assuming that hash indices are chosen independently, the expected false positive rate of our proposed algorithm is $p_f^* = p_{f_L} \times p_{f_A}$, where p_{f_L} is the false positive of the current level and p_{f_A} is the false positive rate of the ancestor level.

Building procedure for the proposed algorithm is exactly the same as for WBSL-BF or LBSL-BF. The search procedure for the proposed algorithm with LBSL-BF is described in the pseudo-code in Fig. 4. The search procedure for WBSL-BF is not presented because of the similarity. In this pseudo-code, one direct ancestor is only queried to verify the positive result of a node.

The proposed algorithm makes more Bloom filter queries because it checks membership for a node and its ancestor as well in the case of a Bloom filter positive in the node. Because the size of a Bloom filter is small enough to fit in an on-chip memory, the time taken by Bloom filter queries can be ignored when compared with the time taken by trie accesses. Furthermore, the proposed algorithm uses exactly the same architecture as the original trie-based algorithm. The decision on whether to check the number of ancestor levels can be adaptively made depending on the false positive rate of a level.

4. Performance Evaluation

Five actual routing tables [8] were used to compare our proposed algorithm with WBSL-BF and LBSL-BF. These routing tables have prefixes roughly from 15000 to 227000, and the number of input IP addresses used in the simulation is three times the number of prefixes in each routing table. A 64-bit cyclic redundancy check (CRC) is used to generate hash indices for both a Bloom filter and a hash table, and a perfect hashing is assumed for the hash table. The size of the Bloom filter for simulation was

Table 1. Performance Evaluation Results with WBSL-BF (BF size: $2F_s$).

				WBSL-BF			The Proposed			
Routing Data	N	N_T	BF size (KB)	No. of BF queries	p_f	No. of trie accesses	No. of BF queries	Scaling factor	p_f	No. of trie accesses
Telstra	227223	452732	128	3255242	0.111	3.514	5650394	1.736	0.085	3.389
Grouptlcom	112310	314986	128	2388570	0.081	3.371	4113662	1.722	0.068	3.311
PORT80	170601	225050	64	1590702	0.136	3.157	2654444	1.669	0.102	3.001
MAE-EAST	39464	172418	64	512900	0.093	2.754	838967	1.636	0.064	2.63
MAE-WEST	14553	76708	32	190281	0.084	2.728	309370	1.626	0.058	2.614

Table 2. Performance Evaluation Results with WBSL-BF (BF size: $4F_s$).

				WBSL-BF			The Proposed			
Routing Data	N	N_T	BF size (KB)	No. of BF queries	p_f	No. of trie accesses	No. of BF queries	Scaling factor	p_f	No. of trie accesses
Telstra	227223	452732	256	3255242	0.026	3.108	5373923	1.651	0.018	3.070
Grouptlcom	112310	314986	256	2388570	0.012	3.047	3947997	1.653	0.009	3.037
PORT80	170601	225050	128	1590702	0.030	2.661	2487263	1.564	0.021	2.616
MAE-EAST	39464	172418	128	512900	0.011	2.401	797129	1.554	0.007	2.383
MAE-WEST	14553	76708	64	190281	0.008	2.395	294848	1.55	0.005	2.382

Table 3. Performance Evaluation Results with LBSL-BF (BF size: $2F_s$).

				WBSL-BF			The Proposed			
Routing Data	N	N_T	BF size (KB)	No. of BF queries	p_f	No. of trie accesses	No. of BF queries	Scaling factor	p_f	No. of trie accesses
Telstra	227223	576370	256	3250168	0.089	3.420	4891309	1.505	0.035	3.134
Grouptlcom	112310	411122	128	2336879	0.132	3.308	3358695	1.437	0.059	2.916
PORT80	170601	299899	128	1586942	0.114	3.069	2211064	1.393	0.048	2.710
MAE-EAST	39464	191757	64	562174	0.088	3.783	969992	1.725	0.062	3.643
MAE-WEST	14553	82156	32	213286	0.111	3.399	316190	1.482	0.061	3.108

Table 4. Performance Evaluation Results with LBSL-BF (BF size: $4F_s$).

				WBSL-BF			The Proposed			
Routing Data	N	N_T	BF size (KB)	No. of BF queries	p_f	No. of trie accesses	No. of BF queries	Scaling factor	p_f	No. of trie accesses
Telstra	227223	576370	512	3250168	0.014	3.066	4760774	1.465	0.005	3.016
Grouptlcom	112310	411122	256	2336879	0.036	2.870	3252403	1.392	0.014	2.755
PORT80	170601	299899	256	1586942	0.019	2.623	2127905	1.341	0.007	2.557
MAE-EAST	39464	191757	128	562174	0.023	3.472	939824	1.672	0.015	3.432
MAE-WEST	14553	82156	64	213286	0.022	2.964	304959	1.430	0.011	2.904

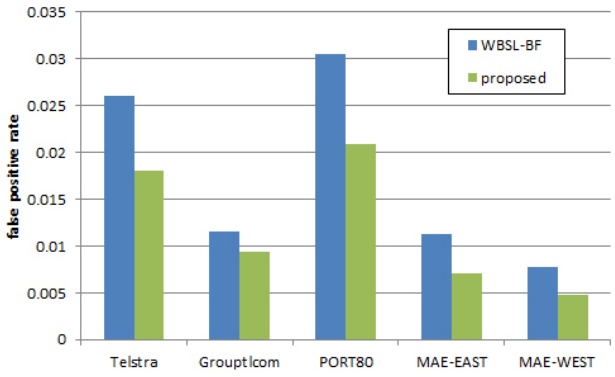

Fig. 5. The comparisons with WBSL-BF in false positive rates.

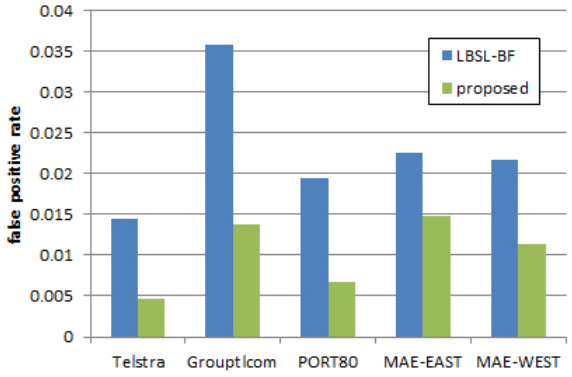

Fig. 6. The comparisons with LBSL-BF in false positive rates.

chosen based on the size factor $F_s = 2^{\log_2(N_T)}$, where N_T is the number of stored nodes. The number of hashing indices k for the Bloom filter was chosen to achieve the minimum false positive rate as shown in Eq. (2).

Tables 1 through 4 show simulation results comparing our proposed algorithm with WBSL-BF or LBSL-BF. In these tables, N is the number of prefixes in each routing table. The number of Bloom filter queries for each algorithm is shown in the table. A scaling factor is the number of queries in our proposed algorithm divided by the number of queries in WBSL-BF (or LBSL-BF). As shown in the scaling factors, our proposed algorithm performs more BF queries in order to reduce the number of false positives.

Figs. 5 and 6 show comparisons in the false positive rates of both algorithms when the BF size is $4F_s$. The false positive rates are significantly reduced, up to 67% with our proposed algorithm. Our proposed method can be applied to various trie-based algorithms with Bloom filters without changing the architectures of the algorithms.

5. Conclusion

This paper proposes a different way of reducing the false positive rate of trie-based algorithms. We proposed to refer the ancestors of a node to confirm that the positive of the node is true. Simulation results show that querying one ancestor of a node can reduce the false positive rate by up to 67% with exactly the same architecture and the same memory requirement. Because there is no change in the architecture itself, our proposal can be adaptively applied depending on the requirement of the false positive rate. Moreover, the proposed approach can also be implemented for other applications with trie-based algorithms employing Bloom filters.

Acknowledgement

This research was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (2014R1A2A1A11051762). This research was also supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2015-H8501-15-1007) supervised by the IITP (Institute for Information & communications Technology Promotion).

References

[1] M. A. Ruiz-Sanchez, E. M. Biersack and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," *IEEE Networks*, vol. 15, no. 2, pp. 8-23, March/April 2001. Article (CrossRef Link)

[2] H. Lim and N. Lee, "Survey and proposal on binary search algorithms for longest prefix match," *IEEE Communications Surverys and Tutorials*, vol. 14, no. 3, pp. 681-697, Third Quarters, 2012. Article (CrossRef Link)

[3] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and Practice of Bloom Filters for Distributed Systems," *IEEE Communications Surveys and Tutorials*, vol. 14, no. 1, pp. 131-155, First Quarter, 2012. Article (CrossRef Link)

[4] S. Dharmapurikar, P. Krishnamurthy, and D. Taylor, "Longest prefix matching using Bloom filters," *IEEE/ACM Trans. Networking*, vol.14, no.2, pp.397-409, Feb. 2006. Article (CrossRef Link)

[5] H. Lim, K. Lim, N. Lee, and K. Park, "On Adding Bloom Filters to Longest Prefix Matching Algorithms," *IEEE Trans. Computers*, vol. 63, no. 2, pp. 411-423, Feb. 2014. Article (CrossRef Link)

[6] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing lookups", *in Proc. of ACM SIGCOMM*, 1997, pp.25-35. Article (CrossRef Link)

[7] J. H. Mun, H. Lim and C. Yim, "Binary search on prefix lengths for IP address lookup," *IEEE Communications Letters*, vol.10, no.6, pp.492-494, June 2006. Article (CrossRef Link)

[8] http://www.potaroo.net

[9] J. H. Mun and H. Lim, "On Reducing False Positives of a Bloom Filter in Trie-Based Algorithms," in Proc. Of ACM ANCS, Oct. 2014. Article (CrossRef Link)

Hyesook Lim received the B.S. and M.S. degrees at the Department of Control and Instrumentation Engineering in Seoul National University, Seoul, Korea, in 1986 and 1991, respectively. She received the Ph.D. degree at the Electrical and Computer Engineering from the University of Texas at Austin, Texas, in 1996. From 1996 to 2000, she had been employed as a member of technical staff at Bell Labs in Lucent Technologies, Murray Hill, NJ, USA. From 2000 to 2002, she had worked as a hardware engineer for Cisco Systems, San Jose, CA, USA. She is currently a professor in the Department of Electronics Engineering, Ewha Womans University, Seoul, Korea, where she does research on packet forwarding algorithms such as IP address lookup and packet classification, and research on content centric networks. She is currently working as the General Affair Director at the Institute of Electronics and Information Engineers. She is a senior member of the IEEE.

Ju Hyoung Mun received the B.S. and M.S. degree at the Department of Electronics Engineering in Ewha Womans University, Seoul, Korea, in 2005 and 2007, respectively. From 2007 to 2013, she was employed at DMC research center of Samsung Electronics, Korea. She is currently pursuing a Ph.D. degree from the same university. Her research interests include packet forwarding algorithms using Bloom filters and forwarding and cache utilization in Content Centric Networks.