Fit the Elephant in a Box - Towards IP Lookup at On-chip Memory Access Speed

Tong Yang*, Alex X. Liu §, Qiaobin Fu†, Dongsheng Yang*, Steve Uhlig¶, Xiaoming Li*.
* Department of Computer Secience, Peking University, China
† Department of Computer Secience, Boston University, USA
§ Department of Computer Science and Engineering, Michigan State University
¶ Queen Mary, University of London, UK

***Abstract*—Fitting large and ever increasing routing tables in small on-chip memory is just like fitting an elephant in a box, which has been considered as impossible. In this paper, we propose the data structure of two-*D*imensional *D*ivision *B*loom *F*ilter (D^{2}BF) that can compactly encode almost all the needed information for performing IP lookup from a FIB in small on-chip memory. With pipelining, we further achieve the throughput of one packet per on-chip memory access.**

I. Introduction

The key challenge of IP lookup is to fit the "elephant" in a "box" - the elephant is the large, yet ever increasing, Forwarding Information Base (FIB, *i.e.*, the IP lookup table), and the box is the small on-chip memory. FIB sizes have been rapidly growing by 15% every year [1], exceeding 600,000 prefixes in 2016 [2]. On-chip memory, such as L2, L3 caches on CPU platforms or Block RAM on FPGA platforms, is inherently small in size (in the scale of tens of Mb) and expensive in price. Although small in size, on-chip memory is best for IP lookup because it is tens of times faster than off-chip memory [3].

Although IP lookup has long been a core networking problem and many types of schemes have been proposed (such as TCAM-based, GPU-based, trie-based, and hash-based schemes), the goal of fitting a large FIB in the on-chip memory and therefore accomplishing IP lookup in the on-chip memory has remained quite distant [4]–[7]. Bloom Filters (in short BFs) [8], as space-efficient data structures, have the potential to compactly encode the FIB information needed for IP lookups in the on-chip memory. Dharmapurikar *et al.* proposed to use Bloom filters in the on-chip memory to encode a FIB [9]; however, their Bloom filters can only be used to find the length of the longest matching prefix in the FIB for a given IP address. After querying the Bloom filters in the on-chip memory, their scheme must access off-chip memory to find the next-hop of the given IP address.

Inspired by this line of thought, in this paper, we propose two-Dimensional Division Bloom Filters (DDBF, in short D^{2}BF). Contrary to Dharmapurikar *et al.*'s scheme, D^{2}BF can be used to directly find the next-hop from a simultaneous query of many Bloom filters. The key insight behind D^{2}BF is that after the Bloom filter query, a single Bloom filter reports true, which points to a single next-hop. Therefore, with D^{2}BF, we do not need to explicitly know the longest matching prefix and therefore do not need to store it on-chip, saving the precious on-chip memory. Furthermore, using the techniques from [10], D^{2}BF can achieve one on-chip memory access per lookup, and almost no off-chip memory access.

II. D^{2}BF

A. Set Lookup Algorithm

Given a FIB $\mathbb{F}$, we build a trie $\mathbb{T}$, then carry out leaf pushing to eliminate overlaps, followed by level pushing to generate a trie with limited number of levels: l_1, l_2, ..., l_L. In this way, we divide the FIB into $L * H$ sets $S_{i,j}(i = l_1, l_2, ..., l_L, 0 < j \leqslant H)$, where i represents the level (prefix length), j represents the next-hop, L represents the number of levels after pushing, and H is the number of next-hops. After this two-dimensional division, the prefixes are divided into L*H sets. Obviously, as L increases, the total number of prefixes decreases and the number of sets increases.

Here we give a formal description of **Set Lookup Algorithm**: After the two-dimensional division, there are L*H sets: $S_{i,j}(i = l_1, l_2, ..., l_L, 0 < j \leqslant H)$. Given an IP address a, we check whether $|a| >> (32 - l_1)$ is an element of $S_{l_1,j}\,(0 < j \leqslant H)$, where $|a|$ represents the integer value of a and $>>$ means "right SHIFT". At the same time, we check whether $|a| >> (32 - l_2) \in S_{l_2,j}\,(0 < j \leqslant H)$, ..., $|a| >> (32 - l_L) \in S_{l_L,j}\,(0 < j \leqslant H)$. After these $L * H$ checks, only one set reports true, and the next-hop is the hop ID of the matched set. $h(S)$ denotes the hop ID of set S.

B. Choosing Optimal Number of Levels

One crucial design choice is the number of levels L, as it impacts the total number of prefixes. Values of L of 3 or 4 typically lead to a minimal number of prefixes. When L is 3 and 4, there are $C_{32}^3 = 4960$ and $C_{32}^4 = 35960$ possible combinations of levels, respectively. For IPv4 FIBs, most prefixes are at level 24, making it an obvious level choice. Also, 32 should be chosen as another level too. As there are few prefixes of lengths between 25 and 31, these levels should not be chosen. Finally, no prefixes currently exist with length between 1 and 7, so these levels should not be considered neither. Therefore, as levels 24 and 32 should be chosen, we are left to decide which levels between 8 and 23 should be chosen. In other words, we are left with $C_{16}^1 = 16$ combinations when $L = 3$, and $C_{16}^2 = 120$ combinations when

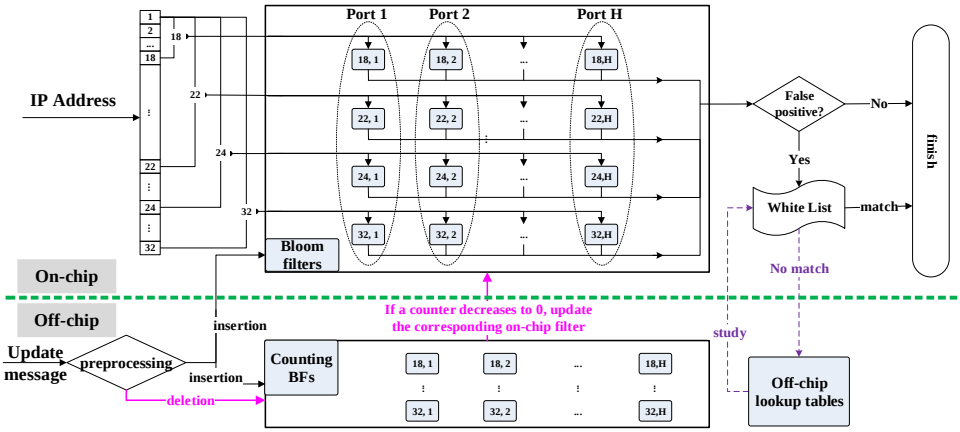

Fig. 1. D^{2}BF architecture.

$L = 4$. Experimental results on 13 FIBs show that: 1) when $L = 4$, the four levels of 18, 22, 24, and 32 achieve the minimum overall number of prefixes in most cases, and 2) when $L = 3$, the three levels of 20, 24, 32 are the optimal levels.

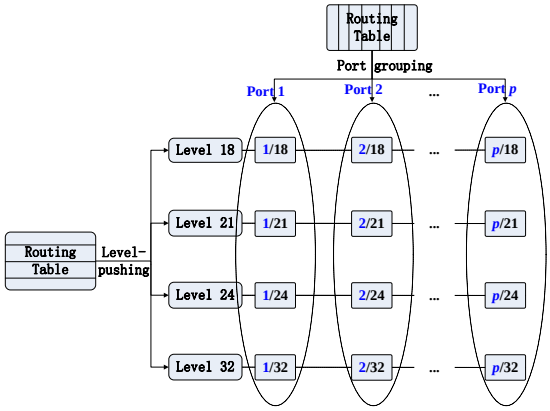

Fig. 2. The construction of TDDBF.

C. Using Bloom Filters

The most important operation in the set lookup algorithm is to judge whether an element belongs to a given set. To achieve this, a naive solution is to traverse the set, with time complexity $O(n)$, where n is the number of elements in the set. Another solution is to construct a hash table, making the average time complexity $O(1)$. However, with the hash table: 1) the worst-case performance cannot be bounded because of hash collisions and; 2) the memory requirements are too large to fit in the on-chip memory if the hash collision probability is to be kept low. Note that perfect hashes cannot be used because of the frequent updates to be made to the FIB. Therefore, in this paper, we propose the use of Bloom filters for the set lookup.

Two-Dimensional Division Bloom Filters: As illustrated in Figure 1, our algorithm works as follows. First, we first build a trie given a FIB. Then we carry out leaf-pushing and level-pushing to produce a trie which has only four levels: 18, 22, 24, and 32 (see Figure 2). We call this trie *4-level trie* in this paper. Second, we build $4 * H$ BFs, named $BF_{i,j}(i = 18, 22, 24, 32;\ j = 1, 2, 3..., H)$, with all $0s$ in all BFs. Third, we traverse this 4-level trie, for each prefix $p/l(p) : h(p)$, where $l(p)$ is p's length (level), and $h(p)$ is p's next-hop. We insert prefix p into $BF_{l(p),h(p)}$, and the construction of the lookup table is complete. The lookup of D^{2}BF is almost the same as the set lookup algorithm. The only difference is that given an IP address a, we perform the BF query to check whether $a >> (32 - i) \in BF_{i,j}$. Assuming that the number of next-hops H does not significantly increase, the lookup speed of the D^{2}BF algorithm, determined by the speed of one query in BF, can be held constant, irrespective of the growth in the routing tables.

ACKNOWLEDGMENT

This work is partially supported by National Basic Research Program of China (2014CB340400), Primary Research & Development Plan of China (2016YFB1000300), NSFC (61672061, 61472184, and 61321491), NSF (CNS-1318563, CNS-1524698, and CNS-1421407), and the Jiangsu High-level Innovation and Entrepreneurship (Shuangchuang) Program.

REFERENCES

[1] X. Meng, Z. Xu, and et al., "IPv4 address allocation and the bgp routing table evolution," *Proc. ACM SIGCOMM CCR*, 2005.
[2] "AS 6447 fib," http://bgp.potaroo.net/as6447/.
[3] W. Feng and H. Mounir, "Matching the speed gap between sram and dram," in *Proc. IEEE HSPR*, 2008, pp. 104–109.
[4] T. Yang, Z. Mi, R. Duan, X. Guo, J. Lu, S. Zhang, X. Sun, and B. Liu, "An ultra-fast universal incremental update algorithm for trie-based routing lookup," in *Proc. IEEE/ACM ICNP*, 2012, pp. 1–10.
[5] T. Yang, G. xie, Y. Li, and et al., "Guarantee IP lookup performance with fib explosion," in *Proc. ACM SIGCOMM*, 2014.
[6] Z. Mi, T. Yang, J. Lu, H. Wu, Y. Wang, T. Pan, H. Song, and B. Liu, "Loop: Layer-based overlay and optimized polymerization for multiple virtual tables," in *Proc. ACM/IEEE ICNP*, 2013.
[7] T. Yang, R. Duan, and et al., "Clue: Achieving fast update over compressed table for parallel lookup with reduced dynamic redundancy," in *Proc. IEEE ICDCS*, 2012.
[8] T. Yang, A. X. Liu, and et al., "A shifting bloom filter framework for set queries," *Proceedings of the Vldb Endowment*, 2016.
[9] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor, "Longest prefix matching using bloom filters," in *Proc. ACM SIGCOMM*, 2003.
[10] Y. Qiao, T. Li, and S. Chen, "One memory access bloom filters and their generalization," in *Proc. IEEE INFOCOM*, 2011, pp. 1745–1753.