A Pipelined Indexing Hash Table using Bloom and Fingerprint Filters for IP Lookup

Heeyeol Yu
Texas A&M University
hyyu@cs.tamu.edu

Rabi Mahapatra
Texas A&M University
rabi@cs.tamu.edu

ABSTRACT

Since achieving a scalable IP lookup has been a critical data path for high-speed routers, such a HT with a predictable lookup throughput for a large prefix table is desirable for these routers. In this paper, we propose a pipelined indexing hash table (PIHT) by using both pipelined BFs in binary search for a key's fingerprint and a fingerprint filter for memory-efficient approximate match. For the IP lookup, a tree-aware prefix collapsing (TPC) converts prefixes with wildcards to collapsed prefixes (CPs) and strides, so that both the PIHT on CPs and bit vectors on strides make a collision-free and perfect $\mathcal{O}(1)$ IP lookup. IP lookup simulation with a large scale BGP table shows that our PIHT offers 3.6 and 7.1 times memory efficiency than other contemporary schemes for 160Gbps routers.

Categories and Subject Descriptors

C.2.6 [**Internetworking**]: Routers

General Terms

Algorithms, Design, Performance

Keywords

IP lookup, Bloom and fingerprint filters, Prefix collapse

1. INTRODUCTION

The demand for high-speed and large-scale routers continues to surge in networking fields. It has been reported that the traffic of the Internet is doubling every two years by Moore's law of data traffic [2] and the number of hosts is tripling every two years [4]. These rapid increases in traffic and hosts lead to two major IP lookup related problems in core routers. 1) Speed: high-speed routers need to look up a routing table at the rate that corresponds to the router's bandwidth requirement. 2) Scalability: a fast IP lookup must be made in searching the longest prefix match with hundreds of thousands of prefixes.

Since a fast packet forwarding is a router's critical data path, literature on packet forwarding has developed three major schemes: Ternary Content Addressable Memory (TCAM), trie-, and hash-based schemes. Unlike TCAM's power inefficiency and trie's unbalanced memory access, hash-based schemes do not perform brute-force lookups and thereby can potentially save an order of magnitude power. Also, hash tables (HTs) use a smaller memory sizes amenable to on-chip memory.

Recently, literature [5, 3] on HTs has focused on their deterministic lookup running at a very low collision rate, so that given a lookup a few off-chip memory accesses are made. These approaches, however, have the following design flaws not suitable for a high-speed and large-scale router as follows: 1) Beyond the generic linked-list implementation limitation, like pointer overhead, an FHT claimed in [5] suffers from two drawbacks: a) duplicate keys in off-chip memory and b) labored *insert* and *delete* operations, both depending on k. A large value of k is not suitable for a high-speed router. 2) Although a Bloomier filter-based hash table (BFHT) [3] utilizes a Bloomier filter [1] and proposes a prefix collapsing (PC), it also inherits two disadvantages of a Blooimer filter: a) a setup failure and b) $\mathcal{O}(n \log n)$ setup and update complexities.

To address these flaws, like prefix duplicates in an FHT and the setup and update failures in a BFHT, we propose a scalable IP lookup hash scheme using BFs and an FF in pipeline. We enhance a PC in [3] to break a prefix into a collapsed prefix (CP) and a tree-aware bit vector. Our scheme, a pipelined indexing HT (PIHT), generates indexes to a key (or CP) table with both BFs and a fingerprint filter (FF) whose are considered as the memory-efficient membership testers. In a PIHT with no pointers, BFs play a role in key search in a b-ary tree $b \in \{2, 4, \cdots\}$, and an FF guarantees the less number of false indexes to a key table in the worst lookup case.

2. A PIPELINED INDEXING HASH TABLE

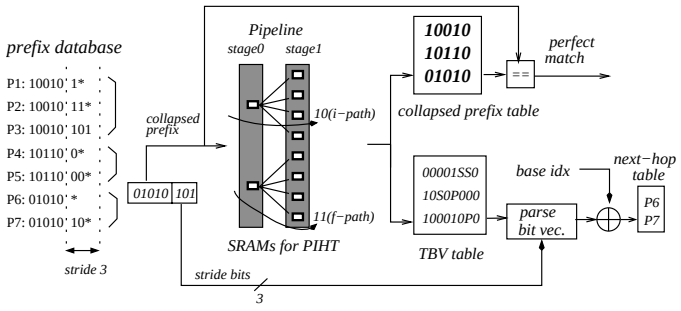


Figure 1: IP lookup architecture with a PIHT, a CPT, and a TBV table

Fig. 1 shows a whole architecture with a PIHT and a TPC for IP lookup. Once the given 7 prefixes are grouped into 3 kinds of

SIGCOMM'08, August 17–22, 2008, Seattle, Washington, USA.
ACM 978-1-60558-175-0/08/08.

collapsed prefixes, three collapsed prefixes and TBVs are stored in a collapsed-prefix table (CPT) and a TBV table, respectively, to be indexed by a PIHT. For instance, an IP address 10110001 in the figure is split into a collapsed prefix and stride bits, and a PIHT is geared to produce an index path (i-path) with possible false paths (f-paths) which are used to index CPT and TBV tables. In the figure, i-path 10 and f-path 11 are generated. CPT entries indexed by i-path 10 and f-path 11 from a PIHT are compared with the packet's collapsed prefix for perfect match. For the NH retrieval, an indexed TBV is parsed to get an index to an NH table. The detailed mechanisms of a PIHT and a TPC are shown in the following.

2.1 Building a Conceptual PIHT

A PIHT for n keys (or CPs) in power of 2 is composed of s=$\log_2 n$ layers (or SRAM module interchangeably) and the index space is partitioned rectangularly of $n \times s$ 0/1 bits, so that a BF covers a column group of the same bits, either 0 or 1, in a key table address space. Fig. 2 shows the PIHT partition where 4 keys are stored in a key table consecutively and the keys' index addresses to the key are partitioned by BFs or an FF on layer j, $0 \le j \le s$-1, resulting each key has its own BF-FF path in the PIHT. In general, n keys are filled in an on-chip key table sequentially from index $0_0 ... 0_{s-1}$ to index $1_0 ... 1_{s-1}$. Let B_j^i denote j-th BF in layer i, $0 \le i \le s$-2 while F_j^{s-1} does j-th FF in layer s-1. Then, if key $e \in S$ is to be inserted at index address A=$a_0 a_1 ... a_{s-1}$, where $a_t \in \{0, 1\}$, $0 \le t \le s$-1, a BF, denoted $B^i_{a_0 \cdots a_i}$ at each layer i, is involved to encode key e just like a legacy BF. In this hierarchical partitioning and encoding, B_j^i covers n_i=$n/2^{i+1}$ keys whose indexes in a key table range from $j \cdot 2^{s-1-i}$ to $(j+1) \cdot 2^{s-1-i}$-1. For instance, B_0^0 and F_3^1 take care of sets $\{e_0, e_1\}$ and $\{e_3\}$, respectively. Although BFs are conceptually partitioned in a layer for their key sets, they concatenate each other in a SRAM module and are separated by their base addresses.

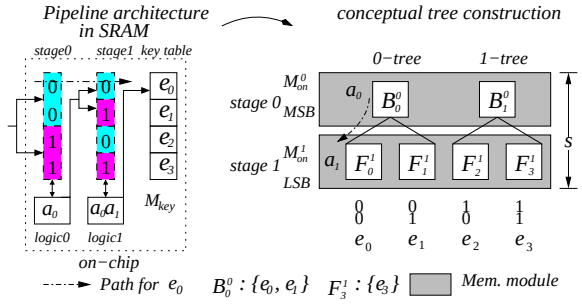


Figure 2: SRAM architecture and a conceptual binary tree construction of a PIHT

2.2 Tree-aware Prefix Collapsing

Any hash-based IP lookup schemes suffer from a wildcard support in a prefix. A PC in a BFHT truncates prefixes into ones of shorter length, so that some prefixes are overlapped and the number of truncated prefixes is smaller than that of old prefixes. However, the number of next-hops is inflated. To remove the next-hop redundancy and to put the next-hop table in on-chip, our TPC uses a TBV which intelligently transforms the tree relationship among prefixes into a vector format using a stack and an index counter.

Fig. 3 shows TPC mechanism with 3 prefixes. The value of stride-sized bits from the packet end indicates how many bits are scanned from a TBV. The TBV's symbols are interpreted according to the symbol table in the figure. If a scanned symbol is '1', the *i_cnt* value is put in a stack, the *i_cnt* is increased by 1 while the value on the top of the stack is an index to an NH table.

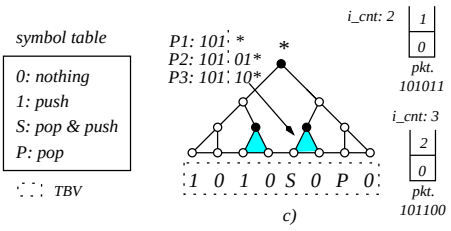
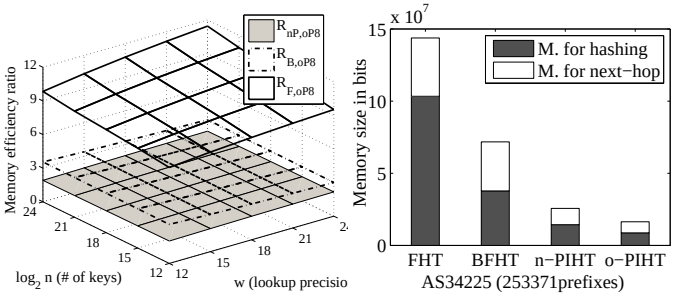


Figure 3: A TPC mechanism with a TBV.

3. EXPERIMENTAL RESULTS

We calculate the memory efficiency ratio among an FHT, a BFTH, and a PIHT to properly address speed and scalability. Fig. 4(a)



(a) Memory efficiency (b) Actual memory size

Figure 4: a) Memory efficiency ratios at various n and w. b) Actual memory size of a BGP table for each scheme, including an HT, a CPT, and a BT.

shows three ratios, $R_{F,oP8}$, $R_{B,oP8}$, and $R_{nP,oP8}$, of o-PIHT(8) over an FHT, a PIHT, and an n-PIHT where an o-PIHT(8) means an optimized PIHT in a 8-ary tree and an n-PIHT denotes a naive PIHT. As shown in this figure, an FHT is absolutely not suitable for speed and scalability concerns. Although $R_{B,oP8}$ at a high lookup precision w is smaller than that of a lower w, it is evident that the overall range of w and n an o-PIHT(8) needs approximately 3 times fewer memory size than a BFHT. Compared to an n-PIHT, an o-PIHT(8) needs about 2 times fewer memory.

In addition to theoretical benefit in Fig. 4(a), we measure the memory size for a BGP table in Fig. 4(b). Note that a dark box in the figure is for memory size of an HT scheme (in our scheme a PIHT and a CPT) while a white box is for a TBV table. In total memory size comparison, an o-PIHT(8) shows 3.6 and 7.1 times better memory efficiency over a BFHT and an FHT on average.

4. REFERENCES

[1] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal. The Bloomier Filter: an Efficient Data Structure for Static Support Lookup Tables. In *SODA '04*.

[2] K. G. Coffman and A. M. Odlyzko. *Internet growth: Is there a "Moore's Law" for data traffic?, Handbook of Massive Data Sets*. Kluwer, New York, New York, 2002.

[3] J. Hasan, S. Cadambi, V. Jakkula, and S. Chakradhar. Chisel: A Storage-Efficient, Collision-free Hash-based Network Processing Architecture. In *ISCA '06*.

[4] Mathew Gray, Internet Groth Summary.

[5] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood. Fast Hash Table Lookup using Extended Bloom Filter: An Aid to Network Processing. In *SIGCOMM '05*.