A Length-Aware Cuckoo Filter for Faster IP Lookup

Minseok Kwon  
Rochester Institute of Technology  
Email: jmk@cs.rit.edu

Pedro Reviriego  
Universidad Antonio de Nebrija  
Email: previrie@nebrija.es

Salvatore Pontarelli  
CNIT  
Email: salvatore.pontarelli@uniroma2.it

Abstract—A cuckoo filter is a highly efficient data structure that provides approximate set-membership testing and addresses some drawbacks of a Bloom filter such as deletion and space overhead. Motivated by the application of Bloom filters to IP lookup, and in general, fast packet processing at routers, we propose a new filter called a length-aware cuckoo filter (LACF), for faster IP lookup with limited extra storage requirement. LACF uses different numbers of hash functions to store and search for entries based on the prefix length popularity of routing entries. Our preliminary results show that LACF reduces the false positive rates for IP lookup by a factor of up to two, with less than 1% storage increase. As future work, we will implement LACF in real network routers, and validate its effectiveness as an IP packet forwarding mechanism.

Index Terms—Cuckoo filters, Bloom filters, IP lookup

I. INTRODUCTION

A cuckoo filter [1] is a compact data structure that can be used for membership test using probability-based hashing as its core technique. It is shown that the cuckoo filter addresses limitations of Bloom filters [2], namely item deletion without filter reconstruction, with no extra space overhead (unlike variants of Bloom filters with deletion support). Bloom filters have been considered to support fast IP lookup and, in general, fast packet processing at switches and routers, as software-based forwarding techniques. Motivated by the practically more enhanced performance, we propose to employ the cuckoo filter for even faster IP lookup with limited additional storage requirement in this work.

We introduce a new filter called a length-aware cuckoo filter (LACF) that treats routing entries discriminatively based on their prefix length popularity. Using different numbers of hash functions based on length popularity is formerly explored by length-aware Bloom filters [3]. Their results discover that false positive rates are reduced significantly (a factor of 4 and 16 for IPv4 and IPv6, respectively), if fewer hash functions are used for the routing entries of popular length (24 in IPv4) while more functions are used for the rest. In the LACF, the routing entries of less popular prefix lengths are inserted into the filter twice, instead of once, to lower false positive rates. This certainly reduces the false positive rates for the addresses of less popular prefix lengths, and almost does not increase the rates for the other addresses. Our analysis and preliminary results show that the false positive rates decrease compared to the case when the cuckoo filter is used.

As proof-of-concept, we will implement LACF in real network forwarding systems, and validate its effectiveness as an IP packet forwarding mechanism. We will first modify the forwarding engine in Linux Ubuntu kernel version 3.13, and then test the algorithm with real routing tables/IP addresses. After successful deployment in the Linux kernel, we also plan to test our implementation of LACF in a Vyatta production system [4]. To evaluate the performance, we will measure and compare the speed, throughput, space complexity, and reliability of our LACF implementation against those of other forwarding techniques.

II. LENGTH-AWARE CUCKOO FILTER

The cuckoo filter inserts a small fingerprint (instead of 0 or 1) of the element in one of the two positions given by hash functions, and subsequently moves the element between those two positions to accommodate new elements inserted later. Figure 1(a) shows an example in which an element $x$ is inserted using two hash functions, $h_1$ and $h_2$.

In LACF, elements are categorized into either popular or unpopular lengths. The basic idea is to use a double insertion for elements that belong to the unpopular lengths to reduce the false positive rate, while an element of the popular lengths is inserted only once as usual. Of course, then searching for an element of the unpopular lengths, which is known a-priori, needs to examine two places in the filter to ensure that the element is present. Interestingly, this approach can reduce the false positive rates for the unpopular lengths, with a negligible impact on the false positive rates for the popular lengths, as shown later. A double insertion is illustrated in Figure 1(b), in which the filter has eight slots to keep an element, and an independent set of hash functions $(h_{10}, h_{20})$ is used for the second insertion.

![Fig. 1. Element insertion in (a) cuckoo filter, (b) double insertion.](image)

The false positive rate for the cuckoo filter is approximately $o\left(\frac{8}{27}\right)$ with eight slots for an element as shown in Figure 1, where $f$ is the number of bits in the fingerprint and $o$ is the percentage of filled positions in the filter, i.e., occupancy. The false positive rates for double insertions can be estimated similarly as $o^2\left(\frac{8}{27}\right)^2$ because the two sets of hash functions must give a false positive and they are independent. This shows...
how the false positive rate is reduced for unpopular lengths. The impact on storage will be low as long as the fraction of elements that are doubly inserted is small. This is the case as double insertion is used only for unpopular prefix lengths.

III. APPLICATION: IP ADDRESS LOOKUP

The analysis of IPv4 BGP tables indicates that approximately 55% of the addresses have the prefix length of 24 bits and most of the rest 45% are 14 to 23 bits [5]. This characteristic makes IP lookup a problem well-suited to LACF. For an incoming IP packet, all possible prefixes of its destination IP address are tested in the filter, and only the longest prefix matched is used for search in the actual IP lookup table [3]. This reduces lookup time significantly as the filter lookup can be performed extremely fast and the number of searches in the actual IP lookup table is limited to only those that return positive by the filter. In this application, the prefixes whose length is smaller than 14 or larger than 24 are unpopular.

For comparison, we implement LACF and the cuckoo filter, and run experiments on a BGP routing table that contains approximately a half million entries collected by the RIPE RCC [6]. In the experiments, the prefixes whose length is smaller than 14 or larger than 24 are doubly inserted, while the other entries are inserted only once. The doubly-inserted prefixes account for a small percentage of the table, and hence the impact on storage is minimal (<1%). We compute the false positive rates as 10,000 packets are tested using different filter sizes over filter occupancy levels 20-90% where the size of a fingerprint is 8 bits.

The results are shown in Figure 2 where the x-axis denotes the percentage of filled elements in the filter and the y-axis denotes the false positive rate. For the results, we first count all the false positives, and then divide the count by the total number of lookups, which is the sum of the number of lookups per packet—each as a packet can be searched multiple times for different prefix lengths. This allows a fair comparison with the theoretical estimate of the false positive rate. In the figure, LACF reduces the false positive rate nearly in half compared to the cuckoo filter (referred to as “Traditional” in the figure) for all the filter occupancy levels. The false positive rates for both the cuckoo filter and LACF increase as the filter occupancy increases as more prefixes in the filter raise chances for false positives. We also observe that the theoretical estimate is close to the false positive rates for the cuckoo filter, and the impact on storage is negligible.

IV. FUTURE WORK

Replacing a routing table parsing algorithm is challenging to field test as vendors are resistant to allowing access and manufacturing/development cycles can be long. However, placing the algorithm in an operating router is critical as simulation and emulation have significant shortcomings. Routing tables possess a number of variables beyond prefix match including administrative distance, metrics and configuration options such as filters which may be difficult to emulate completely.

We propose the following approach: Initially the algorithm will be tested against open source options. The goal is to compile the new algorithm into the code using the Linux forwarding engine. However, in order to provide enough variability to verify results, a large routing table from a production router will be installed in the virtualized open source routers. This actual routing table will ensure that real-world variables are included in the tests. Two options will be tested: a software router [4] implementation and the routing capability built into the Ubuntu distribution. The idea is to install the routes, and compare the default performance against the new algorithm.

As the router forwards traffic, latency across the box will be measured. In the experiments, we will measure the following additional performance metrics: 1) forwarding table lookup time, 2) false positive rates, 3) memory usage, and 4) throughput as the amount of data processed per unit time. Once the baseline measurements have been completed, the modified parsing algorithm will replace the existing and the measurements will be run again. Subsequently a Cisco router will be deployed. This router will have specifications that allow it to verify results, a large routing table from a production router will be installed in the virtualized open source routers. Completed measurements will be compared and evaluated which may lead to modifications to the algorithm.

V. ACKNOWLEDGMENT

This work was supported in part by the Spanish Ministry of Economy and Competitiveness in the frame of the excellence network Elastic Networks TEC2015-71932-REDT and the European Union in the context of the “BEBA” project (Grant Agreement: 644122).

REFERENCES