Performance analysis of a length-aware cuckoo filter in FD.io VPP

Abstract—Packet forwarding is a key operation performed by software routers and hence, this needs to be optimized. The concept of longest prefix matching is used to find the next best hop for a packet by looking up IP prefixes in a routing table. Bloom filters and cuckoo filters have been used to reduce the time taken for IP lookup by eliminating cases of incorrect IP prefix lookups. Length-aware cuckoo filter (LACF) is an improvement of a cuckoo filter which aims to take advantage of the IP prefix length distribution in the routing table. The lookup rate can be increased by reducing the false positive rates associated with the filter.

Index Terms—IP lookup; programmable data plane; packet forwarding; probabilistic filter

I. INTRODUCTION

A network router is a device that forwards packets between and within computer networks. It has routing policies configured in it which contain rules that are to be followed while computing the next hop for a data packet that arrives at the router while on its way to its eventual destination. When an IP packet is created, it consists of a source, and a destination IP address. A router cannot possibly have rules for each individual IP address in the world, hence the rules described in a router span a set of IP addresses, or what are more commonly called IP network prefixes.

A cuckoo filter is a practical and compact data structure that uses a probability-based hashing technique to test the approximate membership of an element in a set. As compared to a traditional Bloom filter, it was proposed to provide the ability to delete elements of a set, and reduce the false positive rate even when it was 95% occupied [5]. A major advantage of the Cuckoo filter was that the addition of these features did not come with an extra overhead on the existing Cuckoo filter.

IP lookup in routers has been traditionally performed using hashes where network prefixes are used as keys. However, the challenge has been with determining the longest matching prefix. This challenge stems from the fact that a routing table in a router might have several matching prefixes for a given destination IP. In that case, the goal is to find the best match, which is referred to as the longest prefix matching. The currently attempted modifications to the lookup, such as tries and Bloom filters, have been implemented with memory footprints small enough to be fit in the cache, which make the computations extremely fast. This paper proposes the implementation of a modified Cuckoo filter, called a length-aware Cuckoo filter [2], that would use a network prefix length and its popularity to add it in the filter discriminately. This approach is directed towards achieving faster IP lookup along with lower false positive rate compared to the performance of the traditional Cuckoo filter.

Cisco has implemented a technique called Vector Packet Processing (VPP), which aims to optimize the utilization of CPU instruction cache for the IP lookup by processing a group of packets together instead of individual processing. A Cuckoo filter and a trie structure have both been individually integrated with the VPP logic and their performances have been documented. This paper aims to evaluate the performance of the length-aware Cuckoo filter when integrated with VPP. The motivation behind trying this modification of the filter is that when such a similar modification was tried out in a traditional Bloom filter [3], there were increased lookup rates seen: an increase by a factor of 4 when looking up IPv4 addresses, and an increase by a factor of 16 when looking up IPv6 addresses.

II. BACKGROUND

In this section, we discuss prior work done with respect to the use of probabilistic filters for faster IP lookup.

A bloom filter [1] is a data structure that can be used to represent the probabilistic presence of a member in a data set. Since the memory footprint of this structure is considerably lower than a traditional hash table, this was initially proposed as a promising way forward.

Park et al. [3] state that although there had not been a significant improvement in the performance of the bloom filters since their inception, there had been several ideas proposed to do so. The authors argued that none of these current modifications to a bloom filter had brought about a significant increase in the efficiency of the filter.

They had observed earlier work which suggested the use of a different number of hashes to reduce the false positive rate in the filters, however, since that required computationally expensive operations on popularity and query distributions,
Park et al. [3] proposed the idea of implementing length-aware bloom filters needing the comparatively less expensive computation of length distribution across the membership sets only.

The authors presented an algorithm to use different number of hashes for various lengths of IP address prefixes. The idea behind their algorithm was a set of popular prefixes would be hashed into the filter using lesser number of hash functions whereas the lesser popular prefixes would be hashed using a larger number of hash functions. They were able to show successfully that their approach decreased the false positive rate of the filter by factors of 4 and 16 for IPv4 and IPv6 addresses prefixes distributed non-uniformly.

Later, Kwon et al. [5] had discussed using a concept similar to the one proposed in [3] over a conventional cuckoo filter due to the advantage of the cuckoo filter being able to handle deletion of entries and having better space complexity as compared to a bloom filter. The authors' proposed algorithm suggests the use of different number of hash functions for various lengths of the IP prefixes that would be used as keys in the filter.

The authors were able to show a reduction of the reported false positives by up to a factor of two whilst only noticing a space overhead increase of 1%.

Kwon et al. [4] proposed the idea of a length-aware cuckoo filter building on top of the suggestions made by Kwon et al. [5]. The authors experimented with the proposed solution by using it with Cisco’s VPP (Vector Packet Processing) for IP lookup and measured the performance of the system by focusing on IPv6 inputs.

The preliminary results show that using VPP with the length-aware cuckoo filter yielded an approximate 31% decrease in the lookup time as compared to the regular VPP operation. The authors attributed this increase in performance to the existence of the compact filter in the cache.

III. IMPLEMENTATION

A conventional cuckoo filter runs an element through a hash function to obtain its fingerprint, which it then inserts in one of two positions given by another two hash functions [5]. Figure 1 illustrates the working of a cuckoo filter.

In a length-aware cuckoo filter (LACF) [2], element lengths are categorized a-priori as being either popular or unpopular. The idea is to then use two hash functions on the unpopular ones, i.e., insert the fingerprints of these elements twice, and use just one hash function on the popular ones, i.e., insert their fingerprints once. When the lookup operation is performed, the unpopular prefix lengths are searched for twice whereas the popular ones are searched for once. This is seen to reduce the false positive rate of these elements while the change in the false positive rate of the popular lengths is negligible. Figure 2 compares the difference in fingerprint insertion in a cuckoo filter (a) and a LACF (b).

Kwon et al. [2] have described the mathematical proof which leads to a lower false positive rate due to the double insertion logic in the LACF. Since there are two different hash functions used for the insertions, each have an independent false positive rate which when multiplied with each other contributes to the improved lookup performance.

Our aim is to analyze the impact of integrating the LACF with VPP. To maximize the performance of the filter lookup, our goal is to ensure that the size of the filter can be controlled so that it can be accommodated in the memory cache. To achieve this, we reduce the size of the routing table by converting a 128 bit IPv6 address to a 16 bit fingerprint. Since the hash computation is a critical part of the lookup process, we ensure that its speed and accuracy are acceptable. We use the xx binary hash function for this calculation. This results in a 32 bit fingerprint of which we utilize the first 16 bits to be inserted into the filter.

We also need to ensure that this filter remains in the cache. This is achieved in VPP by utilizing Linux kernel’s LRU (Least Recently Used) mechanism [6] for inserting and replacing data in cache. Hence, a frequently accessed filter will reside in cache. The design of the implemented solution is such that the filter is invoked for lookup for each incoming packet. This ensures that the filter will be present in the memory cache while packets are being processed.

The integration of the LACF with VPP requires us to modify the existing lookup algorithm by prefixing our filter to the lookup logic. This flow is implemented to perform a FIB table lookup only for those IP prefixes that the filter determined to possibly be a valid prefix for the current packet’s destination IP address. This change also helps in collecting filter statistics as all possible IP prefixes would run through the filter. The filter is first populated with the fingerprints of the available IP routes before the IP route is inserted into the FIB table.

A synthetically generated routing table with 500,000 ran-
randomly distributed routes is used. The routes with prefix lengths /120, /64, /48, and /32 are significantly more popular than the other prefix lengths.

IV. RESULTS

The results were computed by performing 500,000 lookups over filters with varying number of routes. The LACF was fully integrated with VPP and the lookup data was fed to the system using flat files.

The route scales used ranged from 500,000 to 2,000,000.

The results were seen to support the fact that the length-awareness feature of a cuckoo filter leads to an overall better performance not just in terms of lookup rate (speed) but also in terms of false positive rates.

![Fig. 3. Lookup rate for randomly generated traffic](image1)

![Fig. 4. False positive rate for randomly generated traffic](image2)

V. CONCLUSION AND FUTURE WORK

There seems to be quite a drastic improvement in the performance of the length-aware cuckoo filter as compared to the previously implemented cuckoo filter within VPP in terms of false positive rates. Although we expected to see a reduction in this rate, the observed results are really impressive.

In order to provide more concrete proof of concept for this filter, several other scenarios would need to be tested and validated. Larger routing tables, more lookups, and IP prefix distributions from real-world datasets would all need to be run on this system.

This project would also benefit from being run in different software environments, perhaps on servers with other processes running. We could analyze the performance of the filter in such situations by adding some intelligence to the filtering algorithm in order to dynamically alter certain parameters of the filter.

ACKNOWLEDGMENT

I would like to thank Professor Kwon for his continued guidance and support during this project. I would also like to thank Krishna Prasad Neupane for sharing his knowledge about the code base and the entire project.

REFERENCES


