BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2023
http://excel.fit.vutbr.cz

Probabilistic Packet Classification Acceleration on FPGA

Denis Kurka

Abstract

Classifying network packets is a crucial task in networking systems, as it allows for efficient routing and filtering of data. Probabilistic filters are a classification method that uses different techniques to approximate the membership of a packet in a set of rules. This work investigates three algorithms: Bloom, cuckoo, and xor filter. The main aim is to compare the performance of these three methods when implemented as hardware components in FPGA systems. The evaluation criteria include error rate, maximal frequency, and FPGA resource usage. The results indicate that the xor filter outperforms the others regarding error rate, which is superior in any error rate category. The Bloom filter is the fastest option for smaller and quicker components where a higher error rate is tolerable. The cuckoo filter is the most resource-efficient when FPGA logic is the primary concern. These findings contribute to the development of optimized classification systems and provide valuable insights into the possibilities of implementing probabilistic filters in hardware architectures.

xkurka05@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Packet classification is crucial in networking, as it allows the packets to be routed, prioritised or blocked. Furthermore, with the growing amount of data, it becomes increasingly essential to study algorithms that deal with this problem efficiently.

The classification boils down to comparing an element against a set of rules and finding an appropriate match. This is commonly done exactly using various approaches such as linked lists or tree data structures. However, with the ruleset increasing, it may be better to sacrifice perfect matching for better space efficiency. Ideally, this introduced error should only be a false positive; if the match is false, it is undoubtedly not in the ruleset.

One of the most well-known approximate set membership algorithms is a Bloom filter [1]. This algorithm is known for its speed and ability to add a new rule to an already constructed filter. It has many variations [2] and can even be implemented in hardware [3]. Other methods include a cuckoo filter, capable of adding and removing rules [4]. On the other hand, xor probing algorithms, which cannot be changed after construction, can achieve the best memory efficiency [5, 6, 7].

As only the hardware implementation of bloom filters was studied, this work focuses on implementing Bloom, cuckoo and a xor filter in FPGA design. The resulting components should reveal the merits and drawbacks of the presented algorithms, not limited to their construction speed and memory efficiency. But also their resource usage and throughput in hardware design.

The results indicate that when targeting low positive rates and best space efficiency, the xor filter is ahead in all comparisons. However, bloom offers much higher throughput for more miniature rulesets than others, doubling compared to the xor probing. The cuckoo filter uses the least FPGA resources and has a better error rate in the practical range than the bloom filter.

2. Probabilistic filters

The key idea behind probabilistic set membership algorithms is to allow a representation of the ruleset that only stores each rule partially. In other words, the filter becomes lossy. With this, lossy filters can store a much higher number of rules in the same space than conventional approaches.

It is common to use hashing to achieve the probabilistic nature. Filters using hash functions to create the lossy ruleset representation can be categorised into three groups [7]:

- And probing filters: An element is found when all locations match. A Bloom filter is an example of this.
- Or probing filters: An element is found when one of the locations matches. A cuckoo filter is an example of this.
- Xor probing filters: An element is found when a bitwise xor of all locations is a match. A xor filter is an example of this.

This categorisation can be seen on the poster in Figure 1. The cuckoo and xor filters use a shortened representation of input called fingerprints. Which are then stored in memory and later compared using the appropriate probing style. Bloom is a fingerprint-less method, making it much simpler to implement than the others. This work focuses on implementing one filter in each category: Bloom, cuckoo and xor filter.

3. Implementation

A common design was created to test all filters in a hardware environment. This environment is shown in Figure 2. Firstly the FPGA is configured with desired filter and specified parameters. The software part of the system then communicates with the hardware via a PCIe bus decoded to an MI interface. With the prepared filter, the software configurator is presented with a ruleset. Filter parameters are read from the FPGA so that the software can construct a filter model. If the model construction is successful, the configurator will begin data transfer to set all memories in the filter required for operation. After this process, the filter is ready for use.

A UVM verification environment was built to check this solution's hardware design and software. Furthermore, verification was used to simulate filter construction, configuration and performance instead of gathering results from deployed design.

It is important to note that even though Bloom and cuckoo filters are dynamic, the implementations mentioned above do not include these features and treat all filters as static. Also, the implemented algorithms differ from the original, mainly in splitting a shared memory into multiple chunks for a single read access on one memory table, optimising for hardware use. This, however, required substantial algorithm changes and is another contribution of this work.

4. Results

In order to measure the all-around usability of mentioned methods, four evaluation criteria are used: false positive rate, bits per element, maximal frequency and CLB usage of the design.

A false positive rate is used, as a negative match is always accurate. Bits per element represent the ability to compress the ruleset. Both false positive rate and bits per element can be measured from the verification environment. FPGA resources are taken from the synthesis of each design.

In Figure 3, the filters are compared with their most favourable configurations. Every point represents one possible configuration of the filter algorithm. Bloom is represented as a curve, as only the two best configurations are shown: Bloom with 7 and 15 hashes.

Given the results, it is clear that the xor filter is superior to others in any false positive rate category. This is especially visible in the practical range of around 0.3% false positive rate, where the xor filter achieves an exceptional 10 bits per element. Bloom performs better than the cuckoo filter when the false positive rate exceeds 0.2%. Below this boundary, even the more precise 15-hash Bloom fails to match the error rate of the cuckoo filter.

On the other hand, even though the Bloom filter is not the most accurate, the Table 1 shows that it is undoubtedly the fastest, reaching frequencies above 800 MHz in picked configurations. More than double in compared to the xor filter.

One configuration was picked and scaled for every filter type to measure CLB usage. This is shown in Figure 4. We can see that the cuckoo filter is the most efficient regarding FPGA resources, especially in small filter sizes. The xor filter has similar usage to hash-7 Bloom. Moreover, Hash-15 Bloom is around 1.8x more complex than hash-7 Bloom and about 3x more than cuckoo.

The goal was to achieve 100 Gbps and higher speeds since the filter is planned for use in Cesnet projects focused on high-speed networks. According to achieved frequencies, it can handle 100 and 200 Gbps. 400 Gbps is a future work plan and can be achieved with multi-packet processing per clock cycle.

From the findings above, it is clear that the xor filter fulfilled its expectations as the most efficient filter. However, when the false positive rate is not a problem, Bloom can achieve much higher throughput than the others. Lastly, the cuckoo filter is a good alternative when FPGA resources are the primary concern.

References

[1] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, jul 1970.

[2] Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz. Theory and practice of bloom filters for distributed systems. *IEEE Communications Surveys and Tutorials*, 14(1):131–155, 2012.

[3] Takuma Wada, Naoki Matsumura, Ryota Yasudo, Koji Nakano, and Yasuaki Ito. Efficient implementations of bloom filter using block rams and dsp slices on the fpga. *Concurrency and Computation: Practice and Experience*, 33(12):e5475, 2021. e5475 cpe.5475.

[4] Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, page 75–88, New York, NY, USA, 2014. Association for Computing Machinery.

[5] Thomas Mueller Graf and Daniel Lemire. Xor filters: Faster and smaller than bloom and cuckoo filters. *ACM J. Exp. Algorithmics*, 25, mar 2020.

[6] Thomas Mueller Graf and Daniel Lemire. Binary fuse filters: Fast and smaller than xor filters. *ACM J. Exp. Algorithmics*, 27, mar 2022.

[7] Peter C. Dillinger and Stefan Walzer. Ribbon filter: practically smaller than bloom and xor, 2021.