Design and Implementation of a Lightweight Bloom Filter Accelerator for IoT Applications

Hüseyin Aydın Seymen[1], Müştak Erhan Yalçın[2]

[1]Department of Electronics and Communication Engineering, İstanbul Technical University, İstanbul, Turkey
seymenh20@itu.edu.tr

[2]Department of Electronics and Communication Engineering, İstanbul Technical University, İstanbul, Turkey
mustak.yalcin@itu.edu.tr

Abstract

The Internet of Things (IoT) has evolved into a rapidly expanding web of devices surrounding our lives. This interconnected ecosystem facilitates a new level of collaboration and the collection of precious data to be distributed and analyzed. However, as the number of connected devices grows, the volume and variety of generated data increase significantly, necessitating efficient and scalable solutions.

Membership testing is one of the challenging the algorithms used in most of the network applications. To make it fast and lightweight, probabilistic data structures like Bloom Filters are employed. Although variants of this algorithm are proposed in the literature, there are very few addressing low power, low resource implementations with recently developed hashing techniques. Therefore, in this work, we present a practical Bloom filter accelerator design with Murmur3 hash, implemented on a Nexys A7 FPGA board. Then, we test and analyze its performance to identify limitations and explore possible improvements.

1. Introduction

The Internet of Things has enhanced the way we interact with the world, forming a sophisticated environment filled with smart devices. These devices and systems constantly exchange, share, and gather valuable information to enable real-time monitoring, automated decision making, and reveal analytics otherwise hidden. This is why applications ranging from transportation, manufacturing, agriculture, healthcare, infotainment, smart cities etc., are all transforming to benefit from the IoT paradigm.

As with all concepts in the engineering field, IoT has its own trade-offs and challenges in addition to its many advantages. Privacy issues, security concerns, growing complexity, high demand for bandwidth, and many others need to be addressed to unlock its full potential [1].

A significant number of the mentioned applications require fast membership testing to accomplish given tasks without without causing data flow bottlenecks. Identification, tracking, secure communication, attack mitigation, data caching, edge computing are all deploy techniques to achieve it. With the number of devices in the network growing, however, fulfilling this requirement will be much more troublesome considering the battery life, processing power, and memory limitations of IoT devices. Therefore, membership testing should be efficient and scalable to be applicable. This forces designers to use probabilistic data structures like Bloom Filters.

A Bloom Filter is a space-efficient data structure used in applications where approximate answers are acceptable. In exchange for false positive queries, it enables fast testing, substantially reduced memory usage, and relatively low power consumption compared to exact matching techniques like content addressable memories (CAM) suffering from very high resource requirement. It leverages hash functions to randomly distribute signatures of elements in a set. These compact signatures occupy significantly less memory space compared to the original data set. After constructing the filter, membership testing can be performed utilizing the same hash functions. The resulting hash values are used to match with previously recorded signatures in the memory.

In the event of a “match”, all locations pointed by the hash results should contain pre-recorded signatures. If any of these locations is empty, it is considered a “no match”. While “no match” means exactly not being a member, a “match” can sometimes lead to a false membership decision. The likelihood of this mistake can be constrained by adjusting memory size and the number of independent hash functions utilized [2]. Therefore, it’s crucial to establish an appropriate tolerance level for an application. Once this level is determined, fine tuning can be performed on the mentioned parameters to meet specific requirements.

In more detail, as the number of elements and therefore resulted signatures recorded increases, the false positive rate (FPR) increases. Conversely, rise in the number of independent hash functions decreases this probability. Enlarging the bloom filter reduces the probability too. Using more resources in the filter means less failure in general. Thus, it’s essential to strike a balance between resource usage and the failure rate. This trade-off can be analyzed, and optimizations can be performed based on the Equation 1, where p is the false positive probability, k is the number of hash functions, n is the number of elements in the set, and m denotes the size of the bloom filter.

$$p \approx \left(1 - e^{-k*n/m}\right)^{k} \tag{1}$$

Another important factor, affecting the performance of the filter is the choice of hash functions. Hash functions are employed to map elements to calculated positions in the bloom filter. Therefore, this mapping should be uniform and evenly distribute elements throughout the filter. Otherwise, signatures for different elements coincide, leading to collisions and an increase in the false positive probability. Additionally, the function should be fast and scalable. The one having high performance with low power and resource requirements should be selected. Considering all these factors, MURMUR3 hash function is selected due to its superior properties mentioned in [3]

In the literature, various types of Bloom Filters and hash functions are utilized in the context of IoT. For applications like stock management and object tracking, identifying unknown tags is an important problem to overcome. It requires heavy data transmission, so a bloom filter based RFID identification method is proposed in [4]. Also, to protect the privacy of personal data and reduce transfer overhead, a bloom filter based face recognition is proposed for IoT applications in [5]. As it can be seen, reducing network congestion and transferring less data is a topic having a lot of attention. For this reason, caching content in IoT devices is proposed in [6]. Handling content searches in this distributed database relies on bloom filters. Another example among many is a driver monitoring system, exploiting bloom filters to diminish computational complexity [7].

Although the bloom filter is an efficient method, it should be optimized for IoT applications too. To be suitable for edge devices and servers, lightweight implementations with optimized hashing functions should be derived. However, most of the recently published work heavily concentrates on manipulation of false positive probability, application specific improvements or making use of bloom filters to improve algorithms working on high performance, high power servers or FPGAs. Therefore, in this work, we present a counting bloom filter (CBF) accelerator implementation that requires low power and resources. We used the MURMUR3 as the hash function due to the reasons stated previously. We run the design on a board having relatively low resources, a Nexys A7 FPGA board, tested it and analyzed its performance. We demonstrated resource allocation and power consumption of the design for two types of memory used as bloom filter. Then, we focused on comparison and possible improvements that can be applied to achieve better performance without compromising the design goals we presented much.

2. Accelerator Design

The design includes a Microblaze processor, an accelerator IP, an Ethernet MAC, a DDR controller, a QSPI interface, a UART interface, on-chip memories, AXI interconnect and other discrete, serial interfaces commonly required in IoT systems. The Microblaze serves as a general purpose IoT processor, running specific tasks. To efficiently carry out the intended IoT application and maintain high throughput network functions simultaneously, it must be paired with a lightweight accelerator IP. Otherwise, network functions could consume a significant portion of the processing power, potentially hindering the proper functioning of the IoT application. A bloom filter accelerator is a perfect example, addressing this problem, and this is the main reason why it is introduced. In this section, a brief description of peripherals and details of the proposed accelerator are provided.

The Ethernet MAC is responsible for the overall communication of the system. Both the IoT application and the network function rely on data flowing through Ethernet. The DDR memory and the controller serves as a high volume bloom filter as described in subsection 2.1. On-chip memories are utilized for different purposes. One of them is Microblaze instruction and data memory. Another two are utilized to temporarily store data to be processed (dual BRAM). The rest serve as a low volume, high speed bloom filter for the version described in subsection 2.2. The UART is the control and test interface of the system. AXI interconnect interfaces all these units providing a low latency, high speed medium. Also, the QSPI interface is employed to hold the system configuration.

The network function requiring fast membership testing is handled by the accelerator. It has command (AXI Slave Lite), bloom filter(AXI Master), data request(AXI Master), and interrupt interfaces. It includes logic blocks, arithmetic units and three FIFOs to provide data read/write, hashing and address calculation (Figure 1 and Figure 2).

The Control Logic unit is the manager of the IP. It communicates with the processor, gathers necessary parameters and commands. Start command, bloom filter update command, MURMUR3 parameters, number of hash functions, read/write addresses are all registered, and necessary internal signals are generated by this unit. It also provides status and an interrupt when the check or update process is completed. When a start command is issued by the processor, the control logic initiates data read logic to fetch data to be processed. The data read logic sends this data to the MURMUR3 accelerator.

The MURMUR3 accelerator consists of 20 Xilinx DSP blocks running at 100 MHz, and produces hash results on every clock cycle with a 5-cycle internal delay. This block combines XOR, multiplication, addition and logic shift operations to produce these results. Although, there are faster algorithms for multiplication, their logic implementation cannot beat Xilinx optimized DSP resources, which is the main reason for this selection. Also, by optimizing critical paths, this IP can run up to 250 MHz with an 18-cycle internal delay. These IPs can be paralleled without any performance degradation, therefore 12 of them can generate up to 3 GHash/sec for Xilinx Artix 100T which is substantially more than needed for the design.

The MURMUR3 unit stores generated hash values in the Hash Out FIFO. FIFOs in the design are used to pipeline units and handle disturbances in the data flow. When the FIFO starts filling, the Address Calculator logic reads data from it and calculates the bloom filter address to be used in the Bloom Read Logic. The signature is read from either on-chip memory(BRAM) or DDR2 and stored in the Bloom Read FIFO. Simultaneously, the data and the corresponding address is kept in the Bloom Update FIFO for the Bloom Update logic to update the filter, if the bloom update command was issued previously. Afterwards, the Result Write logic fetches the results and sends them back to the address designated by the Control logic and the interrupt is issued to notify the processor.

2.1. DDR based design

In the DDR based design (Figure 1), external DDR2 memory is utilized as the bloom filter. It enables significantly larger bloom filters, which enables the filter to scale. Also, by adding more memory(as much as FPGA supports) to the Printed Circuit Board (PCB) design, the bloom filter size can be further increased. However, random access performance of this design is significantly lower than the design with on-chip memory(OCM). Delays from the memory controller(Mig), AXI interconnect, and DDR2 are aggregated, making random data read so slow. The DDR memory is the main bottleneck, which limits many applications as stated in ([8]).

2.2. On-Chip Memory Based Design

In the on-chip memory based design (Figure 2), instead of DDR memory, internal Xilinx BRAM resources are utilized. Also, to get rid of AXI interconnect overhead, the BRAM memory is

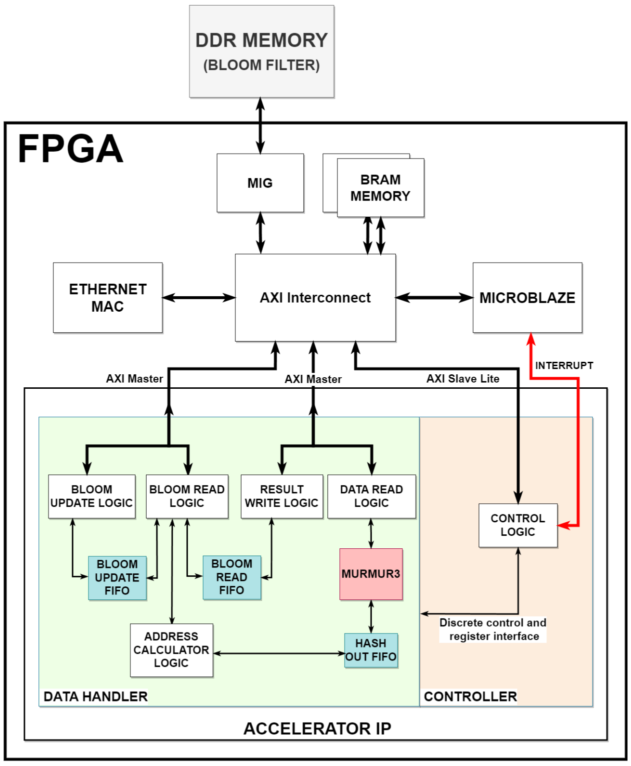

Fig. 1. Block scheme for the DDR based Bloom Filter Accelerator Design

connected to the IP directly. It enables higher bloom filter checking and updating performance which is crucial for network applications in general. Yet, it has limited scalability due to limited FPGA resources. The only option for scaling is having a chip with more internal RAM.

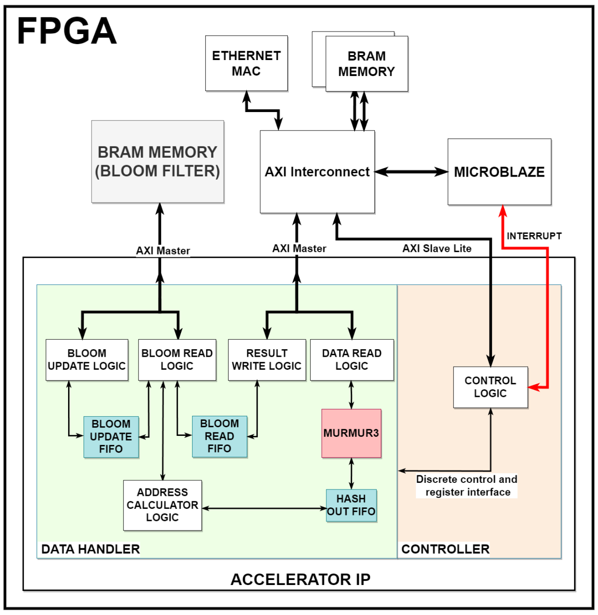

Fig. 2. Block scheme for the On-Chip Memory based Bloom Filter Accelerator Design

3. Results and Discussions

The tests are conducted by filling the data storage dual BRAM (mentioned in section 2) with random 32-bit words. Concurrently, the processor adjusts parameters of the accelerator and sends the start command. Then, the time between the start command and the interrupt is measured by the Microblaze processor. The measurements are collected through UART. Also, using Vivado ILA, exact timings of the IP are measured to ensure and asses its performance. The reason why the Ethernet interface is not utilized as a data source in the tests is speed. The Nexys A7 board has a 100Mb/s Ethernet interface, which is way below what is intended to be tested. In the OCM based design, for 1 hash function, more than 46 MChecks/s is achieved. To provide random 32-bit keys to this system, almost 1.481 Gbps data should be transferred, which is not possible.

The test data is provided in chunks of 16, 32, 64, 128, or 256 keys. It is conducted to reduce effects of overheads in the IP. Having bigger bursts of data makes the IP work more efficiently. This way, the true performance of the IP is achieved which starts saturating with chunks having 256 keys (Figure 3, Figure 4). Therefore, evaluations are conducted with this configuration.

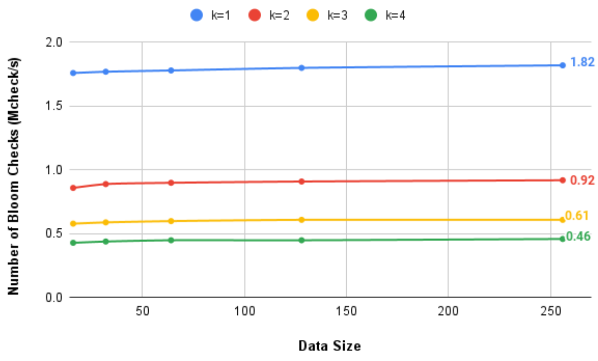

Fig. 3. Effect of Data Size on Performance for DDR Based Design

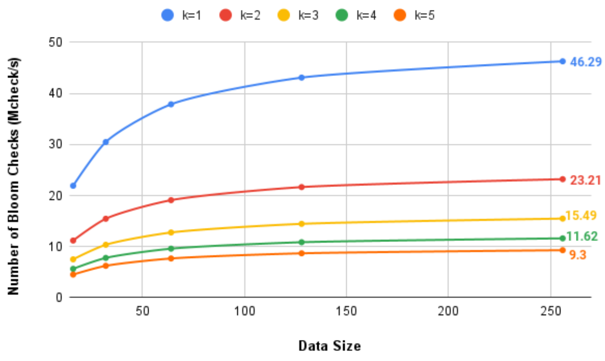

Fig. 4. Effect of Data Size on Performance for On-Chip Memory Based Design

Due to the pipelined design of the units, time spent in MURMUR3 calculation (t_{mm3}), BRAM ($t_{bram\_r/w}$) and FIFO ($t_{fifo\_r/w}$) read/writes contribute to the overall time spent once. The total duration ($t_{bloomcheck}$) is mainly determined by the data chunk size ($data\_size$), bloom filter memory random access delay (t_{bloom_read})

and the number of the hash functions (k).

Equation 2 is an analytic approach to the performance evaluation. Once testing is completed, the results forming the Table 1 are compared with the metrics obtained from Equation 2. The comparison validates the formula as a good representation of the performance.

$$t_{bloomcheck} = k * (t_{bram\_r/w} + t_{fifo\_r/w} + t_{mm3} + data\_size * t_{bloom\_read}) \quad (2)$$

In the DDR based design, having bigger data chunks does not affect the performance (Figure 3) as it does in the OCM based design (Figure 4). Due to the large random access time of the DDR memory, bloom filter read duration dominates the result of the Equation 2, even with very small data sizes. In contrast, when BRAM is used as the bloom filter, random access time is comparable with other delays. Therefore, to achieve optimal performance, data size is increased up to the point where other delays become negligible. This is another detailed reasoning behind using a data chunk size of 256 keys.

Table 1. Performance comparison between DDR based and On-Chip Memory based designs

Number of hash functions	Performance (DDR based) (MCheck/s)	Performance (BRAM based) (MCheck/s)	Acceleration Factor
k=1	1.82	46.29	x25.43
k=2	0.92	23.21	x25.23
k=3	0.61	15.49	x25.39
k=4	0.46	11.62	x25.26

The results shown in Table 1 suggest that the accelerator can perform 1.82 MChecks/s in the DDR based configuration for the number of hash functions, k, equals to 1. Also, the performance is 46.29 MChecks/s in the OCM based configuration which is 25 times better than the DDR based design. Moreover, an increase in the number of hash functions does not affect the acceleration factor while decreasing overall performance. This is mainly because more hash functions correspond to linearly increased number of random memory accesses. These accesses are the dominating factor in the bloom filter checking operation. Therefore, the acceleration factor between designs remains fairly constant while performance drops linearly for both of them.

There are several ways to improve the design, without changing fundamentals of the bloom filter algorithm. Having a 32-bit DDR4-3200MHz instead of 16-bit DDR2 may increase performance up to 12.74 MChecks/s (for k=1, Equation 2) theoretically, for DDR based design. Also, other parts of the design could be improved to further boost the check rate. AXI interfaces can be sped up by increasing the transfer clock rate to at least 200 MHz. Also, increasing bus width from 32bit to 128-bit will result in another 4x acceleration in data transfer between blocks. Additionally, The MURMUR3 IP is currently running at 100MHz and it could be run at 250 MHz as it is described in section 2. Considering all the improvements stated, the DDR based design can perform 17.62 MChecks/s (for k=1, Equation 2) theoretically . Also, with the mentioned improvements, the OCM Based design can perform up to 177.47 MChecks/s (for k=1, Equation 2), which makes it quite preferable.

In the Table 2, a resource allocation comparison is given. The table shows an insignificant increase in resource allocation for the OCM design. At the expanse of requiring slightly more LUTs,

Table 2. Resource allocation values for DDR based and On-Chip Memory based designs

Resource Type	Utilization (DDR Based)	Utilization (BRAM Based)	Available Resource
Slice LUTS	1573 / 2.48%	2641 / 4.17%	63400
Slice Registers	1428 / 1.13%	1835 / 1.45%	126800
BRAM	8 / 5.93%	8 + BF Size / -	135
DSPs	20 / 8.33%	20 / 8.33%	240

registers and BRAM resources, the design can perform 25 times more checks for small bloom filter sizes. For bigger filters, the DDR based design or bigger FPGA should be preferred.

Also, the first two rows of the Table 3 show that the power consumption of the OCM based design is lower due to interface generator IP and external DDR2 memory in the DDR based design. However, for bigger bloom filters, more BRAM resources will be used, which will make overall power consumption closer. Also, it is worth mentioning these values are just for the IP, AXI Interface, utilized memories and controllers, not for the whole IoT System. To make it comparable with the work in [8], consumption of the overall systems are given too (including debug cores).

Bloom filters are utilized in various applications in different forms and types. Therefore, comparing existing works with results presented in incompatible forms, and sometimes, not presented at all, is not straightforward. Although, there is no recent work focusing on resource allocation of bloom filters designed specifically for IoT applications, we compare our work with [8], [9], [10] in Table 3 to evaluate performance achieved.

In [8], the random memory access bottleneck we have in our accelerator IP is targeted. They designed a bloom filter with memory access accelerator. By temporarily storing DDR accesses as their target bank, they manage to read data in bursts which boosts the performance significantly. However, resource utilization and power consumption values are too high in this technique, 175x LUT, 124x BRAM, and more than 8.3x power compared to our DDR based design. It almost fits a bigger, power hungry FPGA, Virtex-7 consuming 25 Watts. Another work having high performance is [9]. With the help of newest generation FPGA, one memory access bloom filter (Bloom-1) and newly generated hash function (Xoodoo-NC), they achieved very high speed operation. On the other hand, with a hash function comparable to the Murmur3 (FNV-1a function), they achieved similar resource allocation and 2x performance compared to our design having similar FPR. However, the improved design, we proposed earlier, potentially manages to double the performance of FNV-1a version even though it involves 4 times more memory accesses. Additionally, they proposed a CAM based exact matching algorithm which outperforms our design compromising resource allocation and possibly power (not provided) which are valuable in IoT applications.

In study [10], they claimed an ad hoc algorithm enabling burst access to DDR memory as it is in [8]. This way they achieved a performance level that falls between the DDR based and BRAM based designs we proposed. To accomplish this burst access characteristic, [8] utilizes lots of resources resulting in a large, power consuming design. However, this is not the case for [10]. Also, they did not provide power consumption value. Therefore, a complete, detailed comparison is not conducted.

Table 3. Comparison with the Studies in the Literature

Work	Technique	Hash Function	FPGA (Xilinx)	Memory	Power (W)	FPR	f (MHz)	Performance (MChecks/s)	LUT / Slices	FF	BRAM	DSP
BRAM Based Design (IP/System)	CBF	Murmur3	Artix-7 100T	On-Chip RAM	0.341 / 2.96	1E7	100	46.29/k (11.58, k=4)	2641	1835	8 + CBF	20
DDR Based Design (IP/System)	CBF	Murmur3	Artix-7 100T	DDR2	1.212 / 2.81	1.3E13	100	1.82/k (0.92, k=2)	1573	1428	8	20
Improved Design (BRAM Based)	CBF	Murmur3	Artix-7 100T	On-Chip RAM	-	1E7	200	177.47/k (44.72, k=4)	>2641	>1835	8 + CBF	20
[8]	BF	Multiple Functions	Virtex-7	DDR3	25W	Not Given	250	372/k (93, k=4)	275739	Not Given	990	Not Given
[9]	Bloom-1	Xoodoo-NC	Virtex UP+	On-Chip RAM	Not Given	2.61E7	462.32	154.1, k=12	675	158	7.5	0
[9]	Bloom-1	FNV-1a	Virtex UP+	On-Chip RAM	Not Given	2.61E7	104.87	26.22, k=12	2706	1562	7.5 + 6	180
[9]	Custom CAM	-	Virtex UP+	On-Chip RAM	Not Given	0	225.07	75.02	20797	988	384	0
[10]	BF	Jenkins Hash	Virtex-5	SRAM	Not Given	4.5E5	150	3.33, k=4	1020 / 450	1210	Not Given	Not Given

4. Conclusions

It can be concluded that the trade off between performance, resources and power should be evaluated with the requirements of applications in mind. Especially for IoT applications, this balance should be adjusted well to achieve optimal performance.

In this paper, we introduced a bloom filter accelerator IP addressing the pressing need for efficient membership testing in the context of IoT. We presented it in two configurations: DDR based and on-chip memory based designs. These configurations aim to enhance the scalability and performance of the design, respectively, while targeting low resources and power installations. We provided test results and analytical analysis of the design's behavior. We compared our work with the studies in the literature and discussed alternative methods to further improve it. Our future plans involve implementing these improvements to assess their real-world impact on performance.

5. References

[1] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1125–1142, 2017.

[2] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 1, pp. 131–155, 2012.

[3] F. Yamaguchi and H. Nishi, "Hardware-based hash functions for network applications," in *2013 19th IEEE International Conference on Networks (ICON)*, 2013, pp. 1–6.

[4] D. Zhang, Z. He, Y. Qian, J. Wan, D. Li, and S. Zhao, "Revisiting unknown rfid tag identification in large-scale internet of things," *IEEE Wireless Communications*, vol. 23, no. 5, pp. 24–29, 2016.

[5] W. Xue, W. Hu, P. Gauranvaram, A. Seneviratne, and S. Jha, "An efficient privacy-preserving iot system for face recognition," in *2020 Workshop on Emerging Technologies for Security in IoT (ETSecIoT)*, 2020, pp. 7–11.

[6] G. Dhawan, A. P. Mazumdar, and Y. K. Meena, "Cncp: A candidate node selection for cache placement in icn-iot," in *2022 IEEE 6th Conference on Information and Communication Technology (CICT)*, 2022, pp. 1–6.

[7] Q. Kong, R. Lu, F. Yin, and S. Cui, "Blockchain-based privacy-preserving driver monitoring for maas in the vehicular iot," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 4, pp. 3788–3799, 2021.

[8] S. Kang, T. S. Ganesh Nerella, S. Uppoor, and S.-W. Jun, "Bunchbloomer: Cost-effective bloom filter accelerator for genomics applications," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*, 2022, pp. 9–16.

[9] A. Sateesan, J. Vliegen, J. Daemen, and N. Mentens, "Novel bloom filter algorithms and architectures for ultra-high-speed network security applications," in *2020 23rd Euromicro Conference on Digital System Design (DSD)*, 2020, pp. 262–269.

[10] P. Lambruschini, M. Raggio, R. Bajpai, and A. Sharma, "Efficient implementation of packet pre-filtering for scalable analysis of ip traffic on high-speed lines," in *SoftCOM 2012, 20th International Conference on Software, Telecommunications and Computer Networks*, 2012, pp. 1–5.