Hindawi
Journal of Computer Networks and Communications
Volume 2019, Article ID 2712417, 10 pages
https://doi.org/10.1155/2019/2712417

Review Article

Hunting the Pertinency of Bloom Filter in Computer Networking and Beyond: A Survey

Ripon Patgiri, Sabuzima Nayak, and Samir Kumar Borgohain

Department of Computer Science & Engineering, National Institute of Technology Silchar, Assam 788010, India

Correspondence should be addressed to Ripon Patgiri; ripon@cse.nits.ac.in

Received 30 August 2018; Revised 6 December 2018; Accepted 1 January 2019; Published 5 February 2019

Academic Editor: Zhiyong Xu

Bloom filter is a probabilistic data structure to filter a membership of a set. Bloom filter returns "true" or "false" with an error tolerance depending on the presence of the element in the set. Bloom filter is used to boost up the performance of a system using small space overhead. It is extensively used since its inception. The Bloom filter has met wide area of applications. Bloom filter is used in entire computing field irrespective of application and research domain. Bloom filter poses (i) high adaptability, (ii) low memory space overhead as compared to hashing algorithms, (iii) high scalability, and (iv) high performance. In this article, we uncover the application area of Bloom filter in computer networking and its related domain.

1. Introduction

Big Data is a disruptive technology in the field of data-intensive computing. Data are generated everywhere. Therefore, the volume of data is growing with an exponential pace, and it continues to grow with the same pace [1, 2]. There are varieties of data generated by various technologies, for instance, IoT. A 90% of these data are unstructured. The high volume of data requires huge memory spaces to process. Therefore, it is prominent to engage the Bloom filter in Big Data. The Bloom filter is like a cog in a machine of a large-scale system. Most importantly, the Bloom filter is used to enhance the performance of a system with a small space overhead. Bloom filters are deployed in various fields to enhance lookup performance. For instance, BigTable uses Bloom filter to eradicate the unnecessary HDD accesses which boost up the lookup performance [3]. Figures 1 and 2 depict the popularity of Bloom filter.

The Bloom filter is used to boost up the lookup performance of a system. It is applied in diverse areas. It improves the system performance dramatically. However, the applications of Bloom filter are limited to membership filter.

The Bloom filter [4] is a variant of hash data structure to implement membership query. The Bloom filter is extensively used and experimented data structures. Therefore, there are numerous variants of Bloom filter, namely, counting Bloom filter (CBF) [5], blocked Bloom filter [6], cuckoo Bloom filter [7], d-left CBF (dlCBF) [8], quotient filter (QF) [9], scalable Bloom filter (SBF) [10], sliding Bloom filter [11], TinySet [12], ternary Bloom filter (TBF) [13], Bloofi [14], BloomFlow [15], difference Bloom filter (DBF) [16], and dynamic reordering Bloom filter [17]. However, a variant of Bloom filters are out of scope in this paper. In this article, we explore the adaptation of Bloom filter in numerous fields. Interestingly, Bloom filters are highly adaptable. Bloom filters are adapted in diverse field, namely, recommendation engine to prevent duplicate recommendation [18], metadata server to enhance lookup performance [19–22], network packet filtering to filter unnecessary data flow [23], IP address lookup [24], network security to prevent malicious user [25–27], Big Data security analytics [28, 29], biometric [30], BigTable [3], P2P [31–33], error correction [34, 35], wireless sensor network [36], duplicate filtering [37, 38], plagiarism checking [39], biomedical engineering [35, 40–42], Web search [43], network router [44, 45], searchable encryption scheme [46], IoT environment [47, 48], spatial Bloom filter [49], in-packet multicasting [50], and databases [51].

The article is organized as follows: Section 2 illustrates Bloom filter. Section 6 discusses on metadata server design using Bloom filter. Sections 7, 4, 8, 3.3, 9.1, 3.2, 8.1, 9, and 5 expose application of Bloom filter in duplicate filtering,

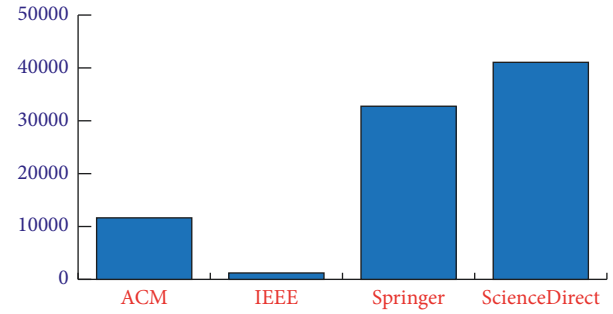

Figure 1: Popularity measurement based on publication of Bloom filter.

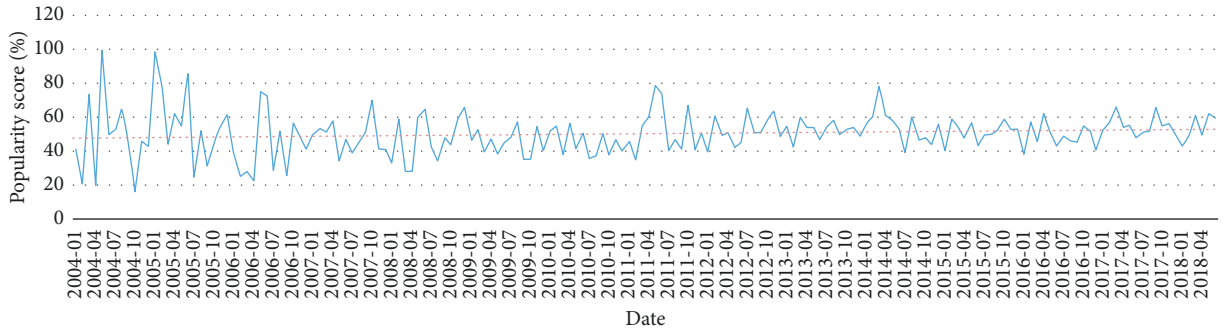

Figure 2: Popularity of Bloom filter worldwide on Google trends.

network security, database, peer-to-peer, wireless sensor network, plagiarism checking, biomedical, and Internet of Things, respectively. Finally, the paper draws a conclusion in Section 10.

2. Bloom Filter

The Bloom filter (BF) [4] introduces an error tolerance to increase lookup performance and space efficiency. The Bloom filter either returns true or false which is illustrated in Figure 3. Thus, the result of Bloom filter belongs to one of the following classes: true positive, false positive, true negative, and false negative. Most of the Bloom filter contains false positive. The false positive introduces overhead to a system. Similarly, a false negative also introduces an overhead for a system. The Bloom filter uses an array to store the information of an element. Let S be a set which is defined as $S = k_1, k_2, k_3, \ldots, k_n$. Let a random query element be k_i where $i = 1, 2, 3, \ldots$. The Bloom filter returns true if $k_i \in S$, otherwise returns false. The false positive is defined as follows: if the Bloom filter returns true when $k_i \notin S$ holds. Similarly, false negative is also defined as follows: the Bloom filter returns false when $k_i \in S$ holds. Thus, the Bloom filter belongs to the probabilistic data structure.

Figure 4 depicts the false positive probability. Bloom filter entirely depends on the number of hash function h. Figure 4 represents the relation among memory $m = 2^{32}$ and number of inputs n and h.

Let n be the total element inserted into the Bloom filter, then, the probability of that bit still 0 is

$$\left(1 - \frac{1}{m}\right)^{nh}, \tag{1}$$

where m is the size of the Bloom filter and h is the total hash function used. Now, the probability of that particular bit to be 1 is

$$\left(1 - \left(1 - \frac{1}{m}\right)^{nh}\right). \tag{2}$$

The probability of all bits becomes 1, which is shown in the following equation:

$$\left(1 - \left(1 - \frac{1}{m}\right)^{nh}\right)^{h} \approx \left(1 - e^{-hn/m}\right)^{h}. \tag{3}$$

The optimal value of number of hash function requirement is

$$h = \frac{m}{n} \ln 2. \tag{4}$$

Number of 1s depends on the number of hash functions h. However, h must be optimal to reduce false positive.

Grandi [52] calculated false positive probability (FPP) through γ – transformations. Let X be the random variable representing the total number set bit in the Bloom filter array, then

$$E[X] = m\left(1 - \left(1 - \frac{1}{m}\right)\right)^{hm}. \tag{5}$$

Let us condition the random variable $X = x$ to determine the false positive, then

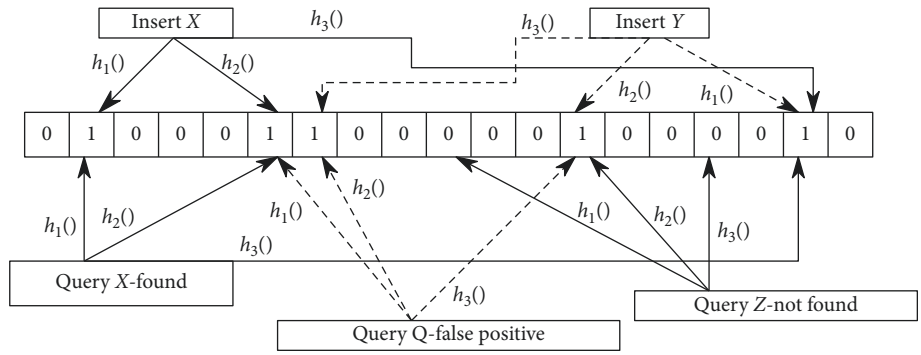

Figure 3: Conventional Bloom filter with $k = 3$ that illustrates the true positive, false positive, and true negative.

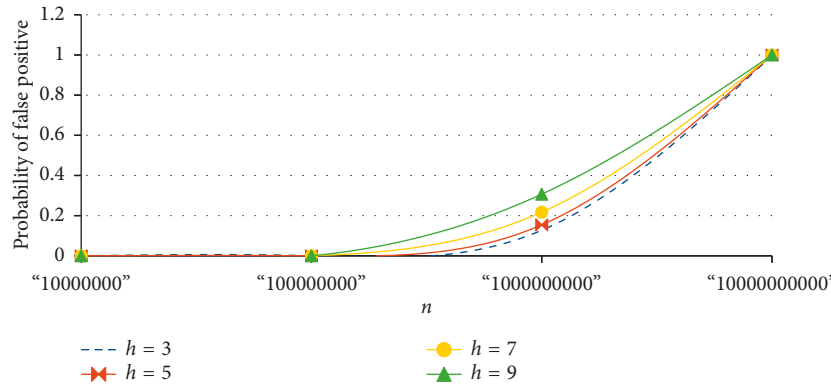

Figure 4: Theoretical probability of false positive. X-axis represents the number of input items, and Y-axis represents probability of false positive in $m = 2^{32}$ memory size.

$$\Pr(\text{FPP} \mid X = x) = \left(\frac{x}{m}\right)^h. \tag{6}$$

Total probability theorems give the exact false positive probability as follows:

$$\begin{aligned}\text{FPP} &= \sum_{x=0}^{m} \Pr(F \mid PX = x)\Pr(X = x) \\ &= \sum_{x=0}^{m} \left(\frac{x}{m}\right)^h f(x),\end{aligned} \tag{7}$$

where $f(x)$ is the probability density function. Grandi [52] performed $\gamma-$transformations to calculate the value of $f(x)$, and the exact false positive probability is given in the following equation:

$$\text{FPP} = \sum_{x=0}^{m} \left(\frac{x}{m}\right)^h \binom{m}{x} \sum_{j=0}^{x} (-1)^j \binom{x}{j} \left(\frac{x-j}{m}\right)^{hn}. \tag{8}$$

2.1. Applications of Bloom Filter. The Bloom filter is deployed in diverse domain to increase performance and reduce memory consumption. The prominent area of Bloom filter's applications is depicted in Figure 5 and also discussed accordingly.

3. Applications in Computer Network

3.1. Network Packet Filter. In the spatiotemporal and secure communication, transmission of duplicate packets is not acceptable. For instance, the duplicate multicasting is a serious issue which is addressed using the Bloom filter [37]. Spatiotemporal approaches are those which convey information within time and with minimum space. It is achieved by maintaining the packet speed during transmission and reducing duplicate packets. The presence of duplicate packets increases the congestion. And, it leads to the overhead in the link and also possibility of attacks in the communication links.

Varshney and Verma [38] proposed an arrangement in which Bloom filter is inserted in the packet header at the time of packet broadcasting. This arrangement helps in elimination of the duplicate packet transmission. In the source node, a Bloom filter is added to the packet header. Initially, the Bloom filter is empty to hide the location of the destination from the attacker. The Bloom filter hashes the packet unique identifier (Pid) and stores it. Then, the packet is transmitted to the neighboring node. When a router or a node receives the packet, it checks whether this packet is received earlier. If yes, then the hash value stored in the Bloom filter checks whether any packet is received with this Pid. If yes, then again, it is checked with unique public key (Pkey) as the Bloom filter, sometime, returns a false positive.

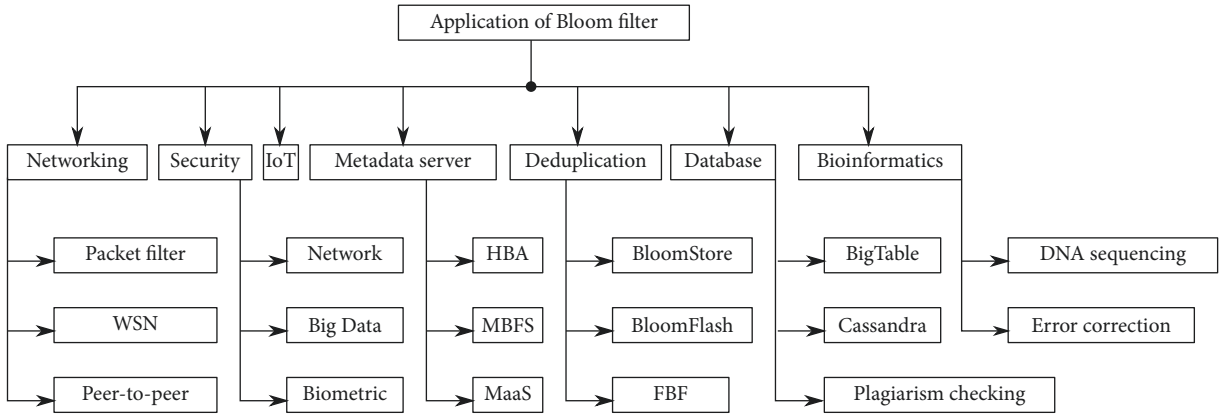

Figure 5: Application of Bloom filter in prominent area.

If it is the case of false positive, then Pkey is stored in the routing table and the packet is transmitted to its neighbor. And, if it is a duplicate packet, then it is discarded. Moreover, if the Bloom filter returns true negative and the packet is received for the first time, then the Pkey value is stored in the Bloom filter, and in the routing table. Then, the packet is transmitted to its neighbor till it reaches its destination node.

Every second millions and millions of packets are generated and transmitted on the network. Hence, the Bloom filter is used for efficient packet filtering techniques. Moreover, it also improves the performance of the network. Fernandez-Del-Carpio et al. propose a hybrid technique [23] where two techniques are combined and shared multicast trees and stateless switching based on Bloom filters (BFs). In this technique, the header of the packet has two fields: one is multipoint label switched (MPLS) path that labels corresponding to the shared tree and the other is the Bloom filter. In addition, every node maintains two forwarding tables: the standard MPLS forwarding table and the link IDs table. In the MPLS forwarding table, each entry is used for shared tree forwarding. And, in the link IDs table, each entry is outgoing interface encoded with Bloom filter. So, when a packet reaches a node, it first checks with the MPLS forwarding table, if successful, then the Bloom filter does the membership checking. This reduces the number of outgoing links to be evaluated and hence reduces traffic overhead.

3.2. Wireless Sensor Network. Wireless network consists of mobile nodes whose location keeps changing. Therefore, a collaborative distributed cache system is proposed to provide efficient services [36]. It is used to store local useful information in a wireless distributed storage system. These storage systems are located in a particular geographic area of interest. It comprises a set of wireless mobile nodes in an infrastructure-less system. And, it permits to use efficiently in numerous location-based applications. However, it is quite difficult to maintain the connection among the stored nodes. Sasaki et al. propose a distributed Bloom filter table (DBFT) [36], which is a two-layered structure overlay. It reduces overhead imposed due to a highly dynamic P2P distributed storage system. DBFT indexes the information packets with their stored nodes using two parameters: cluster-specific Bloom filters and the information packet-specific DBFT weights. Each cluster has its own Bloom filter. In addition, these are periodically exchanged among neighboring clusters. For information packet-specific DBFT weights, a weight vector, size same as the Bloom filter, is calculated using the ID of the stored information packet and set of hash functions. These hash functions are different from hash functions used in the Bloom filter. When an information packet is sent, calculate a sequence of integers from Bloom filter and DBFT weights. Then, it discovers the cluster ID using packet's location. Using this information, the storage node of the packets is discovered. Zhang [53] proposes a simultaneous query technique. This query technique in the wireless network allows access point (AP) to collect key control information from active nodes at low overhead. It uses a typical Bloom filter called analog Bloom filter (ABF). ABF is a Bloom filter that handles analog signals. This Bloom filter is very challenging as the analog signal has random noise and fading. ABF is used in this technique to check whether a node is active or idle. If the ABF returns 1, then the node is active, and if the ABF returns 0, then the node is idle.

3.3. Peer-to-Peer. Peer-to-peer (P2P) is a file-sharing application. It has the potential to become a popular file-sharing protocol. Hence, the performance of the P2P protocol needs to be increased. And, the Bloom filter has the ability to enhance its performance. Chen et al. [31] propose a hybrid P2P network. It uses a gossiping algorithm to gather global statistical information and a Bloom filter in overlay based on DHT global inverted indexes. It uses a Bloom filter for reducing communication cost for multikeyword searching. It is done by adjusting the Bloom filter parameters to optimal setting as per the statistical popularity of the keywords in the query. For a multikeyword searching, a distributed intersection operation is conducted in a wide area network. For this, AND operation is employed. In this operation, first, the query is divided into words, e.g., (X, Y). Second, the Bloom filter is searched for documents containing the first word X. Third, for the second word, intersection of output $\mathrm{BF}(X)$ and document containing the second word Y is found, i.e., $Y \cap \mathrm{BF}(X)$. This process is repeated for all words

in the query. Finally, the Bloom filter output is sent to DHT peer to remove false positive. It is done by finding $X \cap (Y \cap \mathrm{BF}(X))$. Then, this list is sent to the client. Moreover, sometime, the search applies both AND and OR operations for multikeyword searching. In OR operation, first, the query is divided into words (X, Y). Second, the Bloom filter is searched for documents containing the first word X; this list is sent to the client. Third, $Y - \mathrm{BF}(X)$ is also computed and sent to the client.

3.3.1. P2P Traffic Management. Sasaki and Nakao propose a new Bloom filter called queue Bloom filter (QBF) [32], to introduce a new flow classification in P2P. QBF usage reduces the memory consumption of P2P cache. QBF is a new Bloom filter which is a time-series queue. In this Bloom filter, after every fixed time interval, a new Bloom filter is enqueued in QBF. And, when the number of Bloom filter becomes more than the size of the queue, the oldest Bloom filter is dequeued from the QBF. During insertion operation, the new elements are inserted into the new Bloom filter. And, during query operation, all Bloom filters are searched. The flow classification process checks for the flows transmitting duplicate content. This flow classification runs on EGRESS. EGRESS is a filter that monitors and restricts the flow of information from one network to another. Here, it uses QBF for flow monitoring. When EGRESS receives P2P packets, it constructs data pieces. These data pieces are searched in the Bloom filter. If absent, that data piece is inserted into the QBF. If present, then EGRESS counts the number of hits in the flow. If the hit is more than the threshold for addition of a cache entry, then EGRESS knows flow is transmitting duplicate content. Thus, the data pieces are cached. Hence, it removes redundancy in P2P traffic.

3.3.2. Distributed Hashtable. Ariyoshi and Fujita propose a distributed algorithm to process conjunctive queries in P2P DHTs [33]. In this scheme, search results of past queries are cached and used for improving efficiency of the query processing. For this, it uses the Bloom filter. When a requester issues a conjunctive query, a list of peers having that file is obtained. Then, each peer in that list conducts search operation in its Bloom filter. In the Bloom filter, searching for a file is performed based on the index of that file. The Bloom filter returns the list of indices matching the whole conjunctive query or matching some words in the conjunctive query. Lists of indices from all peers are collected, and intersection operation is done to obtain the final result. If a Bloom filter is absent corresponding to the conjunctive query, then a Bloom filter is obtained and multicast to all peers. And, if a peer has a Bloom filter corresponding to the conjunctive query, then it does union operation adding newly added indices. In addition, each peer maintains the Bloom filter using the LRU policy.

4. Security

4.1. Network Security. The Bloom filter is used to provide solution to many network security issues [25, 54]. The Bloom filter is used to address various security issues, and a few solutions have been discussed as follows: Zhu and Mutka [26] propose a message notification protocol. It reduces the power consumption and wireless wide area network access cost for instant messaging. Here, compressed Bloom filter is used to store and represent the message notification exchanged between the IM server and peer group. It provides privacy and security and also reduces overhead of the protocol. Maccari et al. [27] propose an application of the Bloom filter to create a distributed firewall. In this scheme, each node has a Bloom filter which stores the packet accepted by the node. Every node sends their Bloom filter to other nodes. When a node wants to send a packet, it checks with the Bloom filters received. If it matches, then the node sends the packet to the node to whom the Bloom filter belongs to. If the match not found, then the packet is dropped.

4.2. Big Data Security. The crucial point of big data security analytics (BDSA) is the extraction of the value. This extraction becomes more crucial when it is related to threat monitoring and incident investigation to discover both known and unknown cyber attack patterns. In such cases, the Bloom filter is used in many applications related to Big Data security. Furthermore, the Bloom filter is used to profile the network stream and detect malicious patterns. Alsuhibany [28] proposes a Bloom filter called CouBF. The CouBF engages an unsupervised learning engine and also integrates an open digest-based hashing technique [29]. This technique solves the problems in indexing Big Data and BDSA. It counts the occurrences of spam email messages. In other words, a coincidental hit occurs whenever a single cell is used by two or more email messages and increases the counters rapidly.

4.3. Biometric. The Bloom filter is a great data structure for biometric [55–59]. Nowadays, Bloom filter is used to achieve an efficient biometric system. The Bloom filter drastically enhances the performance of the biometric system. The fingerprint matching techniques extract the minutiae points, and these define uniqueness of the fingerprint. The minutiae points of a fingerprint are matched with the fingerprint databases. The fingerprint matching is enhanced using Bloom filter. Moreover, there are numerous such biometric techniques to implement security purpose. For example, iris and face [30]. Numerous works have been done on biometric enhancement; however, the biometric system drastically enhances the performance.

Drozdowski et al. [60] propose an iris indexing scheme using BF. The iris-code template is mapped to a BF. The 2D iris code is separated into a fixed equal sized blocks. In a BF, a single hash function is used for hashing. Using the BF, two iris templates are compared using Hamming distance. Further processing is done using binary search tree.

Rathgeb et al. [57] propose the application of the adaptive Bloom filters (ABFs) to represent the binary iris biometric feature vectors. Use of ABF enables biometric template protection and compression of the biometric data and also speeds up the biometric identification process. The

iris codes are defined in a vector of size $W \times H$. The codes are divided into K equal sized blocks. And, each column consists of $w \leq H$ bits. The entire sequence of the column is transformed to the appropriate location in ABF. So, K number of different ABFs is used. This transform makes the iris codes alignment-free. Furthermore, ABF is parameterized to desired template size. Therefore, this compact alignment-free representation of the iris codes helps in efficient biometric identification.

Sadhya and Singh [55] propose a framework based on a modified Bloom filter. It provides all the desirable security measures required for a biometric template protection scheme. The key idea is perfect secrecy, and it means the encoded data should not leak any information about the original data. In other words, given the encoded data, then a priori distribution of the original data should remain same to its posteriori distribution. In this framework, first construct a key matrix K of size $K \times n$ where K is the number of keys, each having length $n = 2^w$ (w = length of codeword). Each block corresponds to a different Bloom filter. So, there are K Bloom filters of length n. Then, encode these Bloom filters. During identification operation, the encoded Bloom filters are compared with the Bloom filters obtained from features given during authentication. Stokkenes et al. [58] propose a multibiometric template protection based on Bloom filter and binarized statistical image features (BSIF). The features are extracted from face and both periocular regions. The feature vector constructed from these features is given as an input to Bloom filters. For comparison purpose, the Hamming distance between the constructed Bloom filters and the Bloom filters obtained from features given for comparison is calculated. Three different dissimilarity scores are calculated from three different modalities (face, and left and right periocular). This score is analyzed to obtain the final result. Abe et al. [59] propose an irreversible template creation technique using minutiae relation code (MRC) and Bloom filter. In this technique, the Bloom filter is used to define the irreversibility feature. After obtaining the masked MRC, it is given to the Bloom filter to check for its existence.

5. IoT Environment

In the IoT environment, lots of devices are connected to the Internet. They produce a huge volume of data everyday. Among these data, lots of data are not even valuable. Hence, storage and processing of these data consume many times and space. However, in this field, Bloom filter can be very helpful. Singh et al. [47] propose an accommodative Bloom filter (ABF) which helps in insertion of huge data produced by the IoT devices. ABF consists of b buckets of l bits each. Each bucket has a Bloom filter of size $m = m/b$. During insertion operation, hash the new element to p bits. Two hash functions are used, where it uses first q bits for hashing the bucket index and remaining $r = p - q$ bits for hashing the Bloom filter. Whenever a Bloom filter exceeds its capacity, a new Bloom filter is added. During query operation, membership query checking is done at two levels: bucket level and Bloom filter level. This two level checking reduces the query time and enhances the accuracy.

Gundogan et al. propose an optimized low-power routing protocol for IoT devices called BloomRPL [61]. It is an optimized RPL protocol using BF. In BloomRPL, the routers are represented using destination-oriented directed acyclic graph (DODAG). The root of the graph is the sink. The sink is the central control point which gathers information in the IoT network. The BloomRPL uses DODAG Information Solicitation (DIS)/DODAG Information Object (DIO) [62] message handshakes for checking the link between parent and child in the DODAG. The parent stores all the information about its children. This information becomes huge; hence, the BloomRPL uses BF to compress this information. This compression makes the BloomRPL more appropriate for a multicast dissemination. However, use of BF also leads to loss of information and introduces false positive during link checking. But, the merit of using BF exceeds the demerits in BloomRPL.

6. Metadata Server

6.1. Hierarchical Bloom Filter Array (HBA). Hierarchical Bloom filter array (HBA) [19, 20] uses Bloom filter (BF) for metadata management. It uses the Bloom filter in two levels of hierarchy to reduce the memory overhead. At the first level, a small Bloom filter is used to know the destination metadata server (MDS) information. Similarly, in the second level, a pure Bloom array is used which stores the MDS information for all files. For faster lookup, all MDS has a replica of these two Bloom filters.

6.2. Group-Based Hierarchical Bloom Filter Array (G-HBA). G-HBA [63] is an extension of HBA. It is a Bloom filter array that stores the MDS information. This Bloom filter array is used to route directly to the MDS with a high veracity. In this MDS scheme, all the MDS are divided into groups. In each group, every MDS only stores information in its local files and Bloom filter replicas of other groups. So, when information of each MDS in the group is combined, the whole group has the whole file image.

6.3. Multidimensional Bloom Filters (MBFS). The MBFS [21] is a MapReduce-based parallel metadata search technique based on multidimensional Bloom filters (MDBFs). It creates a Bloom filter for each metadata attribute. Then, these Bloom filters are combined to form MDBF. MDBF prunes the subdirectory tree partition which helps in narrowing the search namespace. This leads to fast and accurate search results. Also, in every MDS, MBFS runs concurrently to improve the metadata query efficiency. Additionally, MBFS is lightweight, and when other metadata service execution is done, the query execution is done in MDBF simultaneously. However, it does not provide the deletion operation of metadata. Over a period of time, the Bloom filter becomes exhausted which results in an erroneous output.

6.4. Metadata as a Service (MaaS). MaaS [22] models an efficient MDS to retrieve data from cloud data servers. It uses

a Bloom filter called cloud Bloom filter (CBF) array. CBF consists of global Bloom filter (GBF), local Bloom filter (LBF), and Bloomier matrix filter (BMF). Global Bloom filter (GBF) is used for uploading and downloading of data from the data server. GBF is a single-layered Bloom filter storing information about all MDS in the cluster. During upload, the file name is given as input to GBF, then the GBF hashes the file name, and the produced output gives the location of the MDS. Similarly, the LBF is used during metadata updates. And, the BMF is used for analyzing the metadata.

7. Deduplication

Data deduplication is a prominent research area in a datacenter. An enormous data set consists of duplicate data, and most of the data are unstructured. A huge RAM size is required to process the enormous data silo for deduplication. However, the Bloom filter reduces requirement of huge-sized RAM space. Still, the RAM size is unable to process large set of data. The Bloom filter does not fit in the main memory. Therefore, a secondary storage space is used to achieve the goal. HDD is much slower than RAM. Hence, it is very much time-consuming when the Bloom filter is stored in the HDD. Moreover, the read and write is also very costly. In the Bloom filter, reading and writing requires a few bits. Thus, HDD is not a good option for Bloom filter. The alternative solution is NAND flash memory. Therefore, SSD/flash is used to store the Bloom filter instead of HDD.

BloomFlash [64] implements Bloom filter RAM and flash. Initially, the Bloom filter is implemented in RAM and then the Bloom filter is expanded to flash memory. BloomFlash removes deduplication of key/value. Random reads are instantly allowed to access the flash memory. However, the random writes are buffered to form large chunks and then updated in the flash memory. However, random writes are costlier than random reads. Moreover, reducing random writes increase the performance of BloomFlash dramatically. BloomFlash uses hierarchical structure to maintain the list of filters. Similarly, Lu et al. [65] proposed forest-based Bloom filter (FBF) which works on deduplication based on RAM and flash memory. FBF is further extended in BloomStore to index key/value using SHA [66].

8. Database

The structured databases grow its size over a period of time. The table size becomes a monster. A table size crosses petabytes where the conventional system stops working. Therefore, the Google Inc. developed the BigTable [3] to address this issue. The BigTable is a very large-scale database. It uses a Bloom filter to reduce the disk accesses [3]. Chang et al. [3] claims that using the Bloom filter drastically reduces the number of disk accesses. The BigTable examines the presence of data before accessing the HDD. The BigTable accesses disk if and only if the Bloom filter returns true. Otherwise, the BigTable assumes that the data are not present in the HDD. The HDD takes more times than RAM. Thus, unnecessary time is consumed by disk access where data are not present in the HDD. However, if data are present in the HDD, then an overhead is added by looking up the Bloom filter. But, in a practical scenario, the access overhead of Bloom filter is negligible. Therefore, the BigTable reduces disk access time by deploying Bloom filter, and thus reducing the disk access to enhance the performance.

8.1. Plagiarism Checking. Plagiarism has become a huge problem. With the introduction of new methods to share knowledge, the problem of stealing the written knowledge has also increased. And, comparing with such a large volume of existing document is nearly impossible and also time-consuming. Hence, the Bloom filter has the ability to solve this problem. The matrix Bloom filter (MBF) [39] is used for similar document detection. In this, each row is a standard Bloom filter. The Bloom filter stores all N documents. During the insertion phase, separate the new document into its substring called chunk. Before that, preprocessing of the document is done such as removing stop words and stemming words. After generation of chunks, these chunks are hashed into the corresponding row in the MBF. Then, each chunk is given to k hash functions to generate k bits. These k bits are set to 1 in the MBF. During query operation, the new document is separated into chunks and the same procedure is followed till storage of k bit 1s. Then, compare this row of MBF with each existing document rows. Comparison is done by AND operation. This operation produces a vector of bits. If more 1s are there in the vector, then a new document is copied from another existing document.

9. Biomedical Data

The Bloom filter plays a vital role in biomedical data engineering too. For example, the Genome database of millions of people is very large and unimaginable. A single Genome size is almost 3.2 GB. A millions of such Genome database are unimaginable, and searching a single piece of information takes huge time. Fortunately, the BF gives the results faster than any other existing systems. Therefore, the performance of Genome processing is drastically improved by deploying Bloom filter [40, 41]. Moreover, there are numerous biomedical systems which engage BF. For instance, MRI database. Jackman et al. [40] present an article on ABySS 2.0. It uses the BF for resource efficient assembly of large genomes. Use of BF reduces the overall memory requirements and enables to assemble large genomes on a single machine. It is used to represent the de Bruijn graph (C-DBG). In the BF, the hash functions map each k-mer to a set of slots. This set of slots is referred as bit signature. For traversal operation, to find the path, the BF is repeatedly queried to find the successor. Moreover, BF helps in avoiding duplicate sequences of K-mers. A Bloom filter trie (BFT) [67] data structure is also proposed to store the par genome. It uses colored graph for efficient storage and traversal of genome data. It compresses the colored graph and stores in single data structure. It helps in efficient graph traversal. Recently, Pandey et al. [68] proposed a weighted de Bruijn graph to store genome information. It uses Squeakr which is a k-mer counter. The Squeakr is built using

counting quotient filter (CQF). The CQF is similar to counting BF with many more features. It stores multiset elements. In insert operation, Squeakr reads and parses the input files and then inserts k-mers to a CQF. In query operation, the CQF returns the number of instances of an element currently in the multiset.

Mustafa et al. [69] propose two approaches for compression of a colored graph. One is lossless and the another is lossy compression using BF. The colored de Bruijn graph (cDBG) is constructed using the input sequences and the annotation associated with the k-mers generated from the input sequences. The annotation is represented using a binary matrix. The BF is used to compress the binary matrix. Decouchant et al. [70] proposed a filtering approach using BF to classify the raw genomic data into privacy-sensitive and non-privacy-sensitive information. When an attack is made in a privacy-sensitive region of the genome, more information about the individual is extracted. An approach called long read filtering uses BF to create a dictionary of every (k, i)-sensitive sequence in a genome. The (k, i)-sensitive sequence means that in a k nucleotides sequence, the ith nucleotide is sensitive. The long read filter creates several BFs. The sequence is inserted to BF, and the BF checks the sequence in a sliding window of size k with a previously defined dictionary. If found, one sensitive nucleotide is detected. Similarly, several BFs are used in parallel to increase the throughput.

9.1. Error Correction. In cloud and virtualized environment, data are increasing exponentially. These data are also replicated for load balancing and security purposes. However, it leads to the requirement for fast error correction and data reconciliation, even for very small errors. Motivated by cloud reconciliation problems, a Biff code is proposed by Mitzenmacher and Varghese [34]. Biff codes use invertible Bloom lookup tables (IBLT) [34]. In the IBLT, a key value is inserted into each slot. Moreover, each slot contains three fields: keysum, valuesum, and count. Keysum field contains the XOR of all the keys that is mapped to that slot. Valuesum field contains the XOR of all the values of the keys mapped to that slot. And, count field contains the number of keys mapped to that slot. Biff is used to reconcile differences in data between users. For that, a user constructs its own IBLT and shares that IBLT with the other user with whom it wants to reconcile. In addition, the hash functions are also shared. After receiving the IBLT, the later user deletes the key value from its IBLT that is absent in former IBLT. Moreover, Biff is also used for error correction. Suppose a user sent a message to another user. With the message, it also sent reconciliation information, i.e., IBLT. After receiving the message, the receiver uses the reconciliation information to find the erroneous position of the received data. Furthermore, these operations are speeded up by paralleling them.

10. Conclusion

The Bloom filter is extensively used data structures in a large-scale computing. However, the false positive is an overhead. The false positive rate can be decreased by increasing the size of the Bloom filter. But, the probability of the false positive can never be zero. Thus, there are numerous variants of the Bloom filter to eradicate the problem of false positive to apply specific application. Moreover, there are still rooms to modify the Bloom filter, albeit, the Bloom filter is rigorously experimented. The Bloom filter is adapted in various fields for membership query. For instance, search engine. Besides, there are still numerous fields to adapt the Bloom filter. The Bloom filter is a tiny data structure, yet very powerful. This data structure can change the performance of a system. Thus, studying Bloom filter is worthy irrespective of application area.

Conflicts of Interest

The authors declare that there are no conflicts of interest to disclose regarding the publication of this paper.

References

[1] R. Patgiri and A. Ahmed, "Big data: the v's of the game changer paradigm," in *Proceedings of 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City, IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp. 17–24, Sydney, Australia, December 2016.

[2] R. Patgiri, "Issues and challenges in big data: a survey," in *Distributed Computing and Internet Technology*, A. Negi, R. Bhatnagar, and L. Parida, Eds., pp. 295–300, Springer International Publishing, Cham, Switzerland, 2018.

[3] F. Chang, J. Dean, S. Ghemawat et al., "Bigtable: a distributed storage system for structured data," *ACM Transactions on Computer Systems*, vol. 26, no. 2, pp. 1–26, 2008.

[4] B. H. Bloom, "Space/time trade-o s in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[5] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000.

[6] F. Putze, P. Sanders, and J. Singler, "Cache-, hash-, and space-efficient bloom filters," *Journal of Experimental Algorithmics*, vol. 14, p. 4, 2010.

[7] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: practically better than bloom," in *Proceedings of the 10th ACM Intl. Conf. on Emerging Networking Experiments and Technologies, CoNEXT '14*, pp. 75–88, Heidelberg, Germany, December 2014.

[8] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, *An Improved Construction for Counting Bloom Filters*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[9] M. A. Bender, M. Farach-Colton, R. Johnson et al., "Don'T Thrash: how to cache your hash on flash," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1627–1637, 2012.

[10] P. S. Almeida, C. Baquero, N. Preguica, and D. Hutchison, "Scalable bloom filters," *Information Processing Letters*, vol. 101, no. 6, pp. 255–261, 2007.

[11] M. Naor and E. Yogev, "Tight bounds for sliding bloom filters," *Algorithmica*, vol. 73, no. 4, pp. 652–672, 2015.

[12] G. Einziger and R. Friedman, "Tinyset- an access efficient self adjusting bloom filter construction," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2295–2307, 2017.

[13] H. Lim, J. Lee, H. Byun, and C. Yim, "Ternary bloom filter replacing counting bloom filter," *IEEE Communications Letters*, vol. 21, no. 2, pp. 278–281, 2017.

[14] A. Crainiceanu and D. Lemire, "Bloofi: multidimensional bloom filters," *Information Systems*, vol. 54, pp. 311–324, 2015.

[15] A. Craig, B. Nandy, I. Lambadaris, and P. Koutsakis, "Bloomflow: openflow extensions for memory efficient, scalable multicast with multi-stage bloom filters," *Computer Communications*, vol. 110, pp. 83–102, 2017.

[16] D. Yang, D. Tian, J. Gong, S. Gao, T. Yang, and X. Li, "Difference bloom filter: a probabilistic structure for multi-set membership query," in *Proceedings of 2017 IEEE International Conference on Communications (ICC)*, pp. 1–6, Paris, France, May 2017.

[17] D. C. Chang, C. Chen, and M. Thanavel, "Dynamic reordering bloom filter," in *Proceedings of 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 288–291, Seoul, Korea, September 2017.

[18] S. Wang, Z. Zheng, Z. Wu, M. R. Lyu, and F. Yang, "Reputation measurement and malicious feedback rating prevention in web service recommendation systems," *IEEE Transactions on Services Computing*, vol. 8, no. 5, pp. 755–767, 2015.

[19] Y. Zhu, H. Jiang, and J. Wang, "Hierarchical bloom filter arrays (hba): a novel, scalable metadata management system for large cluster-based storage," in *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pp. 165–174, San Diego, CA, USA, September 2004.

[20] Y. Zhu, H. Jiang, J. Wang, and F. Xian, "Hba: distributed metadata management for large cluster-based storage systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 6, pp. 750–763, 2008.

[21] Z. Huo, L. Xiao, Q. Zhong et al., "MBFS: a parallel metadata search method based on bloomfilters using mapreduce for large-scale file systems," *Journal of Supercomputing*, vol. 72, no. 8, pp. 3006–3032, 2015.

[22] R. Anitha and S. Mukherjee, "'MAAS': fast retrieval of data in cloud using metadata as a service," *Arabian Journal for Science and Engineering*, vol. 40, no. 8, pp. 2323–2343, 2015.

[23] G. Fernandez-Del-Carpio, D. Larrabeiti, and M. Uruena, "Forwarding of multicast packets with hybrid methods based on bloom filters and shared trees in mpls networks," in *Proceedings of 2017 IEEE 18th International Conference on High Performance Switching and Routing (HPSR)*, pp. 1–8, Campinas, Brazil, June 2017.

[24] J. H. Mun and H. Lim, "New approach for efficient ip address lookup using a bloom filter in trie-based algorithms," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1558–1565, 2016.

[25] S. Geravand and M. Ahmadi, "Bloom filter applications in network security: a state-of-the-art survey," *Computer Networks*, vol. 57, no. 18, pp. 4047–4064, 2013.

[26] D. Zhu and M. Mutka, "Sharing presence information and message notification in an ad hoc network," in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom 2003)*, pp. 351–358, Fort Worth, TX, USA, March 2003.

[27] L. Maccari, R. Fantacci, P. Neira, and R. M. Gasca, "Mesh network firewalling with bloom filters," in *Proceedings of IEEE International Conference on Communications (ICC'07), IEEE*, pp. 1546–1551, Glasgow, Scotland, June 2007.

[28] S. A. Alsuhibany, "A space-and-time efficient technique for big data security analytics," in *Proceedings of 2016 4th Saudi International Conference on Information Technology (Big Data Analysis) (KACSTIT)*, pp. 1–6, Saudi Arabia, 2016.

[29] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati, "An open digest-based technique for spam detection," in *Proceedings of 20th International Conference on Parallel and Distributed Computing Systems*, pp. 559–564, San Francisco, CA, USA, 2004.

[30] M. Gomez-Barrero, C. Rathgeb, G. Li, R. Ramachandra, J. Galbally, and C. Busch, "Multi-biometric template protection based on bloom filters," *Information Fusion*, vol. 42, pp. 37–50, 2018.

[31] H. Chen, H. Jin, L. Chen, Y. Liu, and L. M. Ni, "Optimizing bloom filter settings in peer-to-peer multikeyword searching," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 4, pp. 692–706, 2012.

[32] K. Sasaki and A. Nakao, "Packet cache network function for peer-to-peer traffic management with bloom-filter based flow classification," in *Proceedings of 2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 1–6, Kanazawa, Japan, October 2016.

[33] T. Ariyoshi and S. Fujita, "Efficient processing of conjunctive queries in P2P DHTs using bloom filter," in *Proceedings of International Symposium on Parallel and Distributed Processing with Applications*, pp. 458–464, Aizu, Japan, July 2010.

[34] M. Mitzenmacher and G. Varghese, "Biff (bloom filter) codes: fast error correction for large data sets," in *Proceedings of IEEE International Symposium on Information Theory Proceedings*, pp. 483–487, Cambridge, MA, USA, July 2012.

[35] Y. Heo, X.-L. Wu, D. Chen, J. Ma, and W.-M. Hwu, "BLESS: bloom filter-based error correction solution for high-throughput sequencing reads," *Bioinformatics*, vol. 30, no. 10, pp. 1354–1362, 2014.

[36] K. Sasaki, S. Sugiura, S. Makido, and N. Suzuki, "Bloom-filter aided two-layered structured overlay for highly-dynamic wireless distributed storage," *IEEE Communications Letters*, vol. 17, no. 4, pp. 629–632, 2013.

[37] M. Sarela, C. E. Rothenberg, T. Aura, A. Zahemszky, P. Nikander, and J. Ott, "Forwarding anomalies in bloom filter-based multicast," in *Proceedings of IEEE INFOCOM*, pp. 2399–2407, Shanghai, China, May 2011.

[38] T. Varshney and K. Verma, "Rectifying flow of duplicacy using bloom-filter," in *Proceedings of International Conference on Computer, Communications and Electronics (Comptelix)*, pp. 300–304, Jaipur, India, 2017.

[39] S. Geravand and M. Ahmadi, "An efficient and scalable plagiarism checking system using bloom filters," *Computers & Electrical Engineering*, vol. 40, no. 6, pp. 1789–1800, 2014.

[40] S. D. Jackman, B. P. Vandervalk, H. Mohamadi et al., "ABySS 2.0: resource-efficient assembly of large genomes using a bloom filter," *Genome research*, vol. 27, no. 5, pp. 768–777, 2017.

[41] J. H. Ziegeldorf, J. Pennekamp, D. Hellmanns et al., "BLOOM: bloom filter based oblivious outsourced matchings," *BMC Medical Genomics*, vol. 10, no. 2, p. 44, 2017.

[42] G. Holley, R. Wittler, and J. Stoye, "Bloom filter trie: an alignment-free and reference-free data structure for pan-genome storage," *Algorithms for Molecular Biology*, vol. 11, no. 1.

[43] Using Bloom Filters to Refine Web Search Results, https://www.microsoft.com/en-us/research/publication/using-bloom-filters-refine-web-search-results/.

[44] W. Gao, J. Nguyen, Y. Wu, W. G. Hatcher, and W. Yu, "A bloom filter-based dual-layer routing scheme in large-scale mobile networks," in *Proceedings of 26th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–9, Vancouver, BC, Canada, August 2017.

[45] A. Marandi, T. Braun, K. Salamatian, and N. Thomos, "BFR: a bloom filter-based routing approach for information-centric networks," in *Proceedings of IFIP Networking Conference (IFIP Networking) and Workshops*, pp. 1–9, Stockholm, Sweden, 2017.

[46] C. S. Chum and X. Zhang, "A new bloom filter structure for searchable encryption schemes," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, CODASPY '17, ACM*, pp. 143–145, New York, NY, USA, 2017.

[47] A. Singh, S. Garg, S. Batra, N. Kumar, and J. J. Rodrigues, "Bloom filter based optimization scheme for massive data handling in iot environment," *Future Generation Computer Systems*, vol. 82, pp. 440–449, 2018.

[48] L. Yang, Q. Zheng, and X. Fan, "RSPP: A reliable, searchable and privacy-preserving e-healthcare system for cloud-assisted body area networks," http://arxiv.org/abs/1702.03467.

[49] L. Calderoni, P. Palmieri, and D. Maio, "Location privacy without mutual trust: the spatial bloom filter," *Computer Communications*, vol. 68, pp. 4–16, 2015.

[50] I. Nikolaevskiy, A. Lukyanenko, T. Polishchuk, V. Polishchuk, and A. Gurtov, "isbf: scalable in-packet bloom filter based multicast," *Computer Communications*, vol. 70, pp. 79–85, 2015.

[51] A. Goyal, A. Swaminathan, R. Pande, and V. Attar, "Cross platform (rdbms to nosql) database validation tool using bloom filter," in *Proceedings of International Conference on Recent Trends in Information Technology (ICRTIT)*, pp. 1–5, Piscataway, NJ, USA, April 2016.

[52] F. Grandi, "On the analysis of bloom filters," *Information Processing Letters*, vol. 129, pp. 35–39, 2018.

[53] Z. Zhang, "Analog bloom filter and contention-free multi-bit simultaneous query for centralized wireless networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2916–2929, 2017.

[54] R. Patgiri, S. Nayak, and S. K. Borgohain, "Preventing DDos using bloom filter: A survey," *ICST Transactions on Scalable Information Systems*, vol. 5, no. 19, article 155865, 2018.

[55] D. Sadhya and S. K. Singh, "Providing robust security measures to bloom filter based biometric template protection schemes," *Computers & Security*, vol. 67, pp. 59–72, 2017.

[56] J. Bringer, C. Morel, and C. Rathgeb, "Security analysis and improvement of some biometric protected templates based on bloom filters," *Image and Vision Computing*, vol. 58, pp. 239–253, 2017.

[57] C. Rathgeb, F. Breitinger, C. Busch, and H. Baier, "On application of bloom filters to iris biometrics," *IET Biometrics*, vol. 3, no. 4, pp. 207–218, 2014.

[58] M. Stokkenes, R. Ramachandra, M. K. Sigaard, K. Raja, M. Gomez-Barrero, and C. Busch, "Multi-biometric template protection: a security analysis of binarized statistical features for bloom filters on smartphones," in *Proceedings of Sixth International Conference on Image Processing Theory, Tools and Applications (IPTA)*, pp. 1–6, Oulu, Finland, December 2016.

[59] N. Abe, S. Yamada, and T. Shinzaki, "Irreversible fingerprint template using minutiae relation code with bloom filter," in *Proceedings of IEEE 7th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, pp. 1–7, 2015.

[60] P. Drozdowski, C. Rathgeb, and C. Busch, "Bloom filter-based search structures for indexing and retrieving iris-codes," *IET Biometrics*, vol. 7, no. 3, pp. 260–268, 2018.

[61] C. Gündogan, C. Adjih, O. Hahm, and E. Baccelli, "Let healthy links bloom: scalable link checks in low-power wireless networks for smart health," in *Proceedings of the 6th ACM International Workshop on Pervasive Wireless Healthcare*, pp. 11–16, ACM, Paderborn, Germany, July 2016.

[62] T. Winter, *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*, 2012.

[63] Y. Hua, Y. Zhu, H. Jiang, D. Feng, and L. Tian, "Supporting scalable and adaptive metadata management in ultralarge-scale file systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 4, pp. 580–593, 2011.

[64] B. Debnath, S. Sengupta, J. Li, D. J. Lilja, and D. H. C. Du, "BloomFlash: bloom filter on flash-based storage," in *Proceedings of 31st International Conference on Distributed Computing Systems*, pp. 635–644, 2011.

[65] G. Lu, B. Debnath, and D. H. C. Du, "A forest-structured bloom filter with flash memory," in *Proceedings of IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–6, Denver, CO, USA, May 2011.

[66] G. Lu, Y. J. Nam, and D. H. C. Du, "Bloomstore: bloom-filter based memory-efficient key-value store for indexing of data deduplication on flash," in *Proceedings of IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–11, Monterey, CA, USA, January 2012.

[67] G. Holley, R. Wittler, and J. Stoye, "Bloom filter trie: an alignment-free and reference-free data structure for pan-genome storage," *Algorithms for Molecular Biology*, vol. 11, no. 1, p. 3, 2016.

[68] P. Pandey, M. A. Bender, R. Johnson, and R. Patro, "debgr: an efficient and near-exact representation of the weighted de bruijn graph," *Bioinformatics*, vol. 33, no. 14, pp. i133–i141, 2017.

[69] H. Mustafa, I. Schilken, M. Karasikov, C. Eickhoff, G. Ratsch, and A. Kahles, *Dynamic Compression Schemes for Graph Coloring*, 2018, bioRxiv, 239806.

[70] J. Decouchant, M. Fernandes, M. Voelp, F. M. Couto, and P. Esteves-Verissimo, "Accurate filtering of privacy-sensitive information in raw genomic data," *Journal of Biomedical Informatics*, vol. 82, pp. 1–12, 2018.