eBF: An Enhanced Bloom Filter for Intrusion Detection in IoT

Fitsum Gebreegziabher Gebretsadik (fitsumg2007@gmail.com)
Mekelle University
Sabuzima Nayak
National Institute of Technology
Ripon Patgiri
National Institute of Technology

Research Article

Keywords: Bloom Filter, Intrusion Detection System, IoT, Big Data

Posted Date: September 2nd, 2022

DOI: https://doi.org/10.21203/rs.3.rs-2012857/v1

RESEARCH

eBF: An Intrusion Detection System using Bloom Filter in IoT

Fitsum Gebreegziabher Gebretsadik[1,2??], Sabuzima Nayak[2] and Ripon Patgiri[2]

Abstract

Intrusion detection is essential to identify malicious incidents and continuously alert many users of the Internet of Things (IoT). The constant monitoring of events generated from many devices connected to the IoT and the extensive analysis of every event based on predefined security policies consumes enormous resources. Accordingly, performance enhancement is a crucial concern of intrusion detection in IoT and other massive Big Data Systems to ensure a secure environment. Like many Big Data systems, the intrusion detection system of the IoT needs to employ the fast membership filter, Bloom Filter, to quickly identify possible attacks. Bloom Filter is an admiringly fast and space-efficient data structure that quickly handles elements of extensive datasets in small memory space. However, the trade-off between the query performance and the number of hash functions and between memory space and False Positive Probability remain issues of Bloom Filter. Thus, this article presents an enhanced Bloom Filter (eBF) that remarkably improves memory efficiency and introduces new techniques to accelerate the search processing demand of intrusion detection systems in IoT. We experimentally show the efficacy of eBF using a real intrusion detection dataset. The experimental result shows that the proposed filter is remarkably memory efficient, faster, and more accurate than the state-of-the-art filters. eBF requires 15x, 13x, and 8x less memory compared with Standard Bloom Filter, Cuckoo filter, and robustBF, respectively. Therefore, this new system will significantly impact the enhancement of the performance of intrusion detection of IoT that concurrently monitors several billion events crosschecking with the defined security policies.

Keywords: Bloom Filter; Intrusion Detection System; IoT; Big Data

1 Introduction

Internet of Things (IoT) refers to the massive network of integrated "Things" on the Internet. The "Things" refers to the physical objects that vary from simple household objects to sophisticated industrial tools embedded with mechanical and digital machines, computing devices, sensors, and people or animals provided with biochip [1, 2]. IoT is a combination of many synchronized technologies that efficiently support the day-to-day activities of human beings [3].

IoT interlinked 11.3 billion devices by 2021, and forecasts show that the number will increase to 29 billion by 2030 [4]. This massive interaction handles Big Data and supports every human activity. The rapid growth of IoT connectivity and ubiquitous technology integration creates a conducive ecosystem for the applications of smart cities, logistics, transportation, health care, and home appliances. However, as the network interlinks heterogeneous sources with correlating events, the risk of

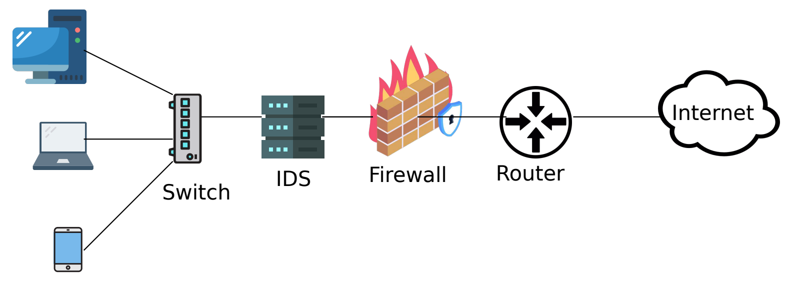


Figure 1 Intrusion Detection System

exposure to intruders becomes high [5]. So, ensuring the security of IoT network from attackers is a significant concern for IoT development [1, 2, 6].

To protect the IoT from attackers, researchers have developed several intrusion detection systems (IDSs) [7]. Figure 1 depicts the basic model of IDS. The IDS monitors every event in the network and analyzes it as per the predefined possible indications of security policy threats or violations. Intrusion is prevented by determining the sign and stopping the detected incident. IDS plans to notify network users about the incoming attacks by continuously monitoring the network traffic. Nevertheless, the vast amount of data accessed and analyzed according to defined signs needs advanced techniques that enhance processing efficiency. The fast and compact membership data structure- Bloom Filter [8]- supports an efficient intrusion detection process providing a true or false match based on hashed bits [9]. Bloom Filter is well known for supporting several Big Data processing systems [10].

Bloom Filter is a space-efficient data structure implemented to boost the performance of searching an element in an extensive dataset using small memory space with high speed. Bloom Filter applies bit-wise data representation, consuming low memory space to handle a large number of queries of Big Data applications. Besides, Bloom Filter implements hash functions to generate a separate digest for every element representation efficiently and uniquely. So, data entry to and data extraction from the filter based on the hashing requires linear complexity of time O(1) [11].

Standard Bloom Filter supports insertion and lookup operations. The introduction of variants changes the classical features to allow deletion operation and speed up the processing capacity. Moreover, this progress reduces the memory space requirement to a relatively better size. Nevertheless, using small memory space for a Big Data key representation creates a false positive error. Because Bloom Filter is a probabilistic data structure, it can return a true or false result with a certain probability. The probability of returning a true result for a key that does not exist in the data set is known as false positive probability (FPP). Thus, the advantage of using low memory representation in a bit representation causes a problem of determining a non-existent element as a member of the set. Besides, the trade-off between time and space requires careful remedy. Accordingly, several studies [11, 12, 13, 14, 15] proposed variants of Bloom Filter. Cuckoo filter [16] was introduced to replace the standard Bloom Filter by avoiding the efficiency and accuracy limitations. robustBF [15] is a space and time efficient multidimensional Bloom Filter variant compared to the Standard Bloom Filter, Counting Bloom Filter, and cuckoo filter. robustBF is also more accurate than these filtering variants. So, the introduction of powerful variants makes Bloom Filter an essential performance optimization data structure for Big Data processing.

Intrusion detection system is a central traffic filtering system. However, the connected device also requires intrusion detection system at their end, for instance, smartphones. Bloom Filter can fulfill such demands. Therefore, we propose a model for intrusion detection system that fits on IoT devices. This article presents a new, highly efficient Bloom Filter variant - eBF to support intrusion detection systems. This new system introduces a new method to improve the efficiency of previous Bloom Filter variants significantly. In addition to controlled synthetic datasets, eBF uses real intrusion detection datasets from IoT to test its applicability in intrusion detection for IoT and other major Big Data applications. eBF aims to achieve the following objectives in comparison with standard Bloom Filter [8], Cuckoo Filter [16] and robustBF [15].

- To significantly reduce memory space consumption of intrusion detection system
- To enhance insertion and lookup speed by implementing efficient algorithms
- To preserve the lowest rate of false positive occurrences

Therefore, this article consists of corresponding sections that illustrate the importance and significance of the proposed system. Section 2.1 precisely demonstrates the fundamental features and basic operations of a Standard Bloom Filter. Section 3 discusses related previous studies and explains the relevance of the new system. Section 4 also demonstrates the model of the proposed method by presenting the algorithms. Besides, section 5 depicts the experimental outcome by presenting comparison graphs. Section 6 precisely discusses the distinctive features of the proposed Bloom Filter variant. Finally, section 7 winds up the presentation of this article with conclusive messages.

2 Background

2.1 Bloom Filter

Bloom Filter is a probabilistic data structure for an efficient membership testing of an element from an extensive dataset. Bloom Filter implements bit-wise data representation to avoid fetching the big dataset from permanent storage to memory. So it uses minimal memory space to run millions of search operations on an extensive dataset. Besides, Bloom Filter applies fast and robust hash functions that consume linear complexity of time O(1) [11] to generate representation hashes without or with a bit of collision probability.

2.1.1 Parameters of Bloom Filter Definition

The main parameters on which Bloom Filter's definition and performance depend are memory size *(M)*, number of hash functions *(K)*, and the probability of false positive occurrences (FPP). The maximum limit of FPP can be predefined based on the system's behavior. The increase in the number of elements *(N)* that the Bloom Filter represents directly affects the memory size and the number of hash functions. Equations 1 and 2 deliver the optimal memory space and the number of hash functions required to develop a powerful Bloom Filter.

$$M = -\frac{N\ lnFPP}{(ln2)^2} \tag{1}$$

The increase in K diminishes the time efficiency of the Bloom Filter. The decrease in K also has the probability of increasing FPP. So the optimal number of hash functions is computed as:

$$K = \frac{M}{N} ln2 \tag{2}$$

If the system developer does not predefine FPP, equation 3 computes the maximum false positive errors compromised to ensure the optimal time and memory efficiency of the Bloom Filter.

$$FPP = \left(1 - \left(1 - \frac{1}{M}\right)^{KN}\right)^{K} \tag{3}$$

2.1.2 Bloom Filter Operations

Standard Bloom Filter allows insertion and searching operations. Figure 2 visualizes the architectural model of a Standard Bloom Filter. The insertion function adds a representation of an element to the allocated memory space of the Bloom Filter, and the search function checks the existence of an element representation in the Bloom Filter. Introducing several variants of the Bloom Filter with the intention of efficiency and accuracy enhancement adds various features to the standard Bloom Filter. However, the basic features and operations remain core factors in the enhanced Bloom Filter variants to implement an appropriate computational optimization algorithm for several Big Data Systems.

Next to memory space allocation for the Bloom Filter, every memory unit is initialized to zero. To insert new elements, Bloom Filter uses one or more hash functions. Based on the hash functions' result, the 0 value of the corresponding cells of the filter is changed to 1. Figure 2 shows that three hash functions (h_1, h_2, *and* h_3) are used to insert A, B and C into the Bloom Filter. Similarly, searching employs these three functions to check the existence of A, R, and S. So, both insertion and searching operations of a standard Bloom Filter use the same hash functions. In the case of searching the Bloom Filter, the element is hashed by the hash functions. The corresponding cells of the filter are checked. If all cell values are 1, then the element exits.

2.1.3 Correct Results of Membership Filter

Big Data System employs Bloom Filter to check the existence of an element in the storage. So, the expected output of the Bloom Filter algorithm is either True (confirming the existence of the requested element) or False (denying the existence of the element). Accordingly, from Figure 2 we can observe that *search(A)* returns *true* because A was inserted by *insert(A)*. Moreover, the figure depicts that *search(S)* returns *false* denying the membership of S. This result is a true answer because the cells expected to be hashed by $h_1(S)$ and $h_2(S)$ both hold *0s*, showing that S is not a member of the dataset inserted in the Bloom Filter.

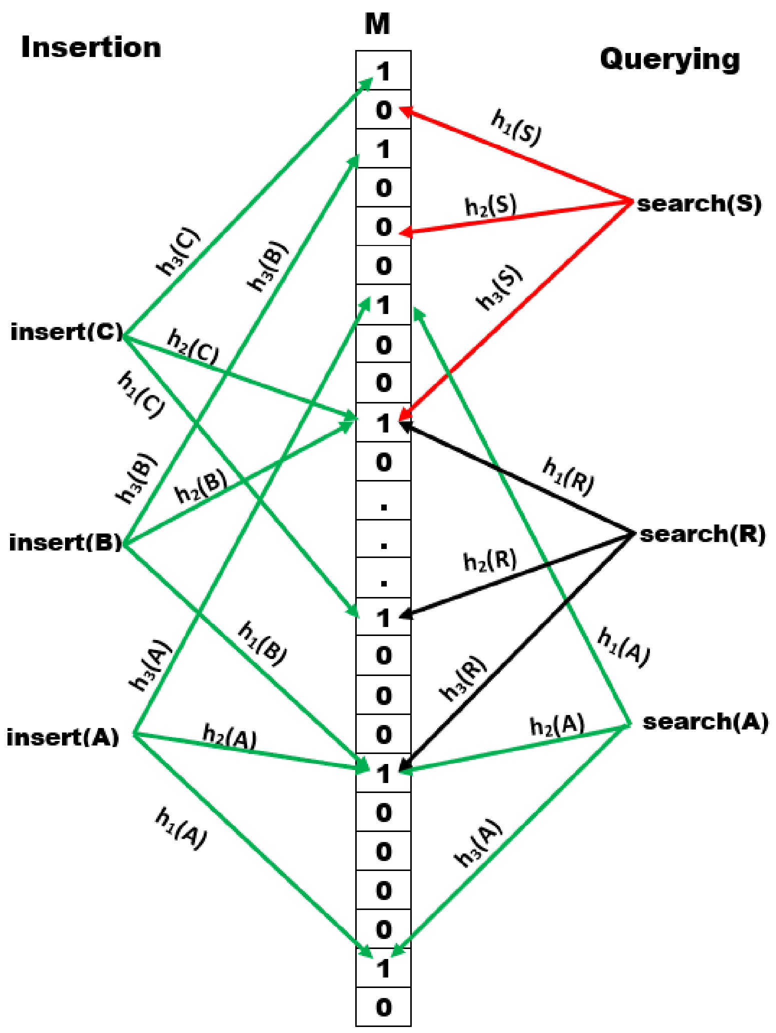


Figure 2 Architecture and operations of Standard Bloom Filter

2.1.4 Probabilistic Data Structure

Probabilistic data structures compromise uncertain answers to lessen the trade-offs between space and time. Probabilistic behavior is relevant for developing customized algorithms per the system's need. Some access control systems may allow visible false positives for account creation to avoid similarities of accounts. Unlikely, customer service that can serve hundreds of millions of clients may strictly diminish false positives near zero to prevent intruders and discard unwanted processes. As a result, using probabilistic data structure has a vital role in enhancing computational efficiency.

Bloom Filter applies probabilistic components to efficiently process the data it holds but cannot provide a definite answer (exactly true/false). Though it is possible to reduce the degree of uncertainty near insignificant impact, the "true" result may not always mean the element is a member of the specified set. Standard Bloom Filter does not have an issue with false negatives. Nevertheless, Unlike the standard Bloom Filter, some variants, such as Counting Bloom Filter, face a probability of denying the availability of an existing element giving out a false negative [17]. Hence, several research works deliver plenty of solution schemes to enhance the accuracy of the algorithms.

2.1.5 False Positive

The main challenge of a Bloom Filter on which many researchers have been working is False Positive [18, 19]. When the number of elements N stored in the bit array with a fixed size M increases, the probability of representing two or more elements using a single bit increases due to hashing collision. Consequently, membership of all the elements hashed to the same location M_i will always reply true for the existence of the not inserted element. That is why the name of this problem is known as False Positive. In Figure 2, *search(R)* returns *true* confirming the presence of R. However, the result is incorrect because all 1's hashed in the cells corresponding to the hash functions of element R ($h_1(R)$, $h_2(R)$, and $h_2(R)$) map representation of elements from A, B and C not representation of element R.

Consequently, in addition to enhancing time and memory efficiency, minimizing FPP is the main issue of studies related to Bloom Filters. The formula to find optimal FPP based on N number of elements inserted to an M size array of Bloom Filter based on K different hash functions is equation 3.

2.1.6 Hash Function

Hash function digests a data input to its compressed size to significantly enhance the processing performance. Cryptographic techniques implement hashing algorithms to secure data. However, non-cryptographic hash functions, including JenkinsHash and MurmurHash, are more time-efficient than cryptographic hash functions [20, 11]. Accordingly, Bloom Filter algorithms prefer implementing non-cryptographic hash functions because cryptographic hash functions reduce the processing speed and do not reduce the false positives [21]. For instance, the Bloom Filter in Figure 2 uses three hash functions ($h_1,\ h_2,\ and\ h_3$) to insert and to lookup an element to and from the Bloom Filter respectively. Hashing helps the Bloom Filter to handle Big Data of complex systems concisely.

2.1.7 Performance and Accuracy Trade-off

Bloom Filter must always deal with how to overcome the competitive challenge between efficiency and accuracy. The accuracy of a Bloom Filter degrades when FPP increases. Increasing the number of hash functions to insert an element is a remedy for reducing FPP. However, increasing the number of hash functions increases computational overload and consumes more memory space, negatively affecting the critical advantage of using a Bloom Filter in small memory space. Hence, a better system that can balance this trade-off without significant negligence on performance and accuracy is the dominant research topic on Bloom Filter. Hence, this article introduces an extraordinarily efficient and accurate Bloom Filter that can enhance the processing performance of Intrusion Detection in IoT and other Big Data Applications.

3 Related Works

Intrusion detection system using Bloom Filter is emerging due to the memory footprint used by the Bloom Filter. It is a fast and memory-efficient data structure. Accordingly, Artan *et al.* [22] proposed a variant of Bloom Filter known as Aggregate Bloom Filter to support network intrusion detection system efficiently. In a similar intention, Brindha *et al.* [9] designed a Counting Bloom Filter to reduce the memory and time used to handle the monitoring process of the network intrusion detection system.

Groza and Murvay [23] implemented Bloom Filer on an intrusion detection system to identify potential attacks in the controller area network that monitors and reports a large number of traffic attacks. The result shows that Bloom Filter is a vital tool for effectively handling intrusion detection processes within a constrained resource. Besides, Bala *et al.* [24] demonstrated the significance of using Bloom Filter to efficiently handle the massive amount of spam observed in SMTP sessions. This work used an intrusion detection system to detect spamming bots of SMTP sessions related to the social network users of a university campus. Zinkus *et al.* [25] designed an intrusion detection system that employs Bloom Filter to handle fuzzy anomaly detection in IoT efficiently. The evaluation of the system shows that the detection of simulated attacks is well enhanced. Hence, developing an enhanced Bloom Filter that supports an intrusion detection system to defend the security of IoT from attackers is crucial.

Intrusion Detection is a highly active research area with several implications. Studies show that intrusion detection has considerable economic impact [5]. A Bloom Filter in intrusion detection aims to determine if a given event's data (e.g., network packet) is part of the predefined set of threats stored in the database. A standard Bloom Filter uses bit arrays to store the representations of the set of threats of the IoT. The hash result of the predefined threats is used as indices of the bit array to set the value of the memory unit to 1. So, intrusion detection systems implement the Bloom Filter to efficiently and effectively identify the possible security attacks of the IoT. Hence, using an enhanced Bloom Filter is very important to protect the billions of devices linked to the IoT from malicious attacks [26, 23].

The employment of the classical data structures to efficiently and effectively handle the immensely increasing size of data in cyberspace is insufficient [27]. Consequently, various advanced techniques and algorithms have evolved for decades. Probabilistic data structures are among the advanced data structures highly relevant to help the applications that manage Big Data [28]. Probabilistic data structures are essential approaches to resolve the challenge of high latency due to the extremely high data processing in systems like cloud computing, financial systems, and social media that simultaneously engage billions of users' interactions. Probabilistic data structures use non-encryption hash functions, including murmur hash, to facilitate the access of elements in a lower memory space [11]. So the advantage of using probabilistic data structures to handle the searching process is more critical in terms of time and space efficiency when compared with the traditional structures. However, the issue of delivering inaccurate results remains a trade-off with the time and space advantages. So, the probability of error occurrences needs minimization to an insignificant level. Big Data Systems widely employ Bloom Filter and its variants as time and space efficient probabilistic data structure [29, 30]. The intrusion detection systems of IoT and other complex Big Data applications use Bloom Filter because it is a pertinent approach for representing sets and supports efficient search execution.

Lucchesi *et al.* [31] employed Bloom Filter to design an optimized IP lookup algorithm. Because of Bloom Filter, the proposed algorithm highly optimizes the throughputs of the IP lookup. The above studies justify that Bloom Filter is an important data structure to handle Big Data and efficiently support Intrusion Detection of networks. However, the different systems implement the Standard Bloom Filter, which requires enhancements.

Bloom Filter is helpful in approximation representation. However, similar to other probabilistic data structures, it faces the issue of false positives. Thus, several studies proposed improved schemes to:

1 Reduce the memory space required to represent relatively huge data economically.
2 Increase the performance of search processing by implementing high-speed algorithms.
3 Lessen the false positive rate to an insignificant degree or near zero.

Counting Bloom Filter [32] is among the variants of Bloom Filter that uses a counter bit in addition to the representation bit. Unlike the standard Bloom Filter, counting Bloom Filter allows deletion operation. Every corresponding counter increments or decrements when an element is inserted into or deleted from the filter. Harwayne-Gidansky *et al.* [26] presented an intrusion detection system based on Counting Bloom Filter (FPGA SoC) to achieve a scalable and high degree of throughput. Despite its advantage of including deletion of elements for some applications that require deletion, counting Bloom Filter introduces memory overhead and consumes more time complexity to process [28, 13]. In addition, Counting Bloom Filter exercises a high degree of an FPP that degrades the system's accuracy.

CountBF [13] enhances the time and memory efficiency of standard Bloom Filter and counting Bloom Filter, preserving low FPP. A r-multidimensional Bloom Filter (rDBF) [14] is proposed, which has a significantly fast filtering algorithm with lower memory space and fewer false positives. This scheme introduced a new view of hashing that requires the X and Y coordinates to minimize the trade-off between memory space and FPP. Unlike the Standard Bloom Filter, this multidimensional variant avoids the dependency on the number of hash functions. The decrease in the number of hash functions without degrading the accuracy quality boosts the processing speed.

The introduction of Cuckoo Filter [16] was to replace the use of the Bloom Filter. Mosharraf *et al.* [10] used Cuckoo Filter to enhance the searching performance of distributed Big Data Systems. The proposed scheme doubled the performance of the searching in the targeted Big Data clusters. Cuckoo Filter has time, and memory efficiency advantages over the Standard Bloom Filter [8]. However, elements can get rid of the insertion queue and be placed in an alternative bucket as a result that increases insertion time. The rapid growth in size and complexity of Big Data Systems demands a continuous engagement in performance improvising methods.

robustBF [15] is the 2-dimensional feature of rDBF [14] is incorporated with the modified murmur hash function to enhance the processing speed and ensure high

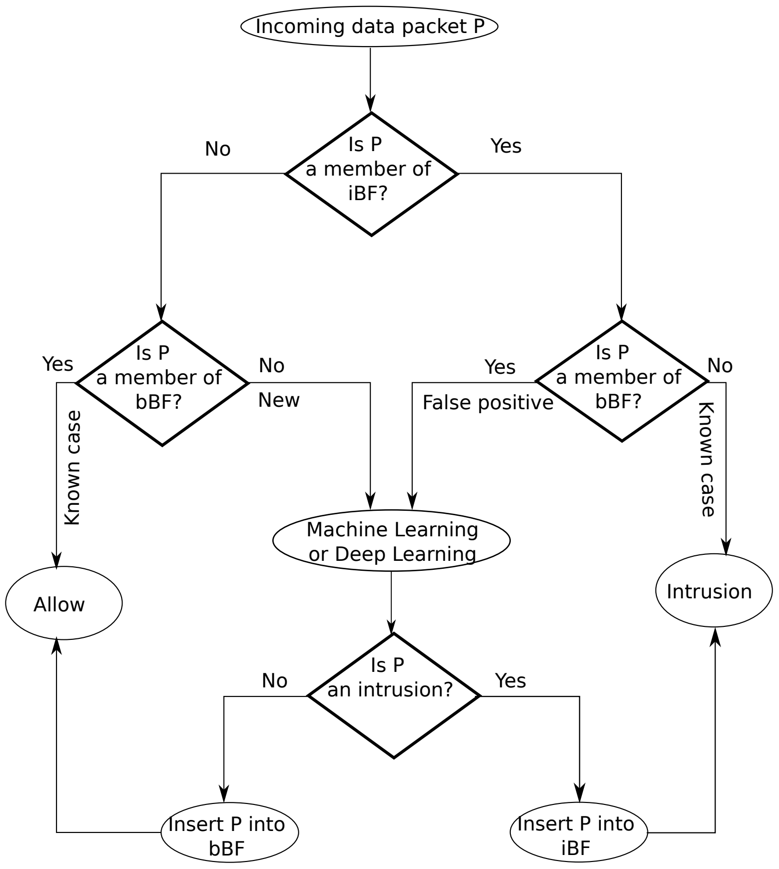


Figure 3 Flow-chart of our proposed system for IDS.

accuracy, diminishing the FPP near zero. The article demonstrates that robustBF uses at least 10 and 44 times lower memory space than standard Bloom Filter and Counting Bloom Filter, respectively. Nevertheless, memory consumption needs enhancement for better efficiency. Besides, the insertion and search speed requires improvement to cope with the rapid growth of the IoT Domain. The intrusion detection systems used in the IoT also require an enhanced Bloom Filter that can accelerate the threat determination to serve the fast-growing number of devices.

Conclusively, the exponential growth of data size and complexity of IoT and similar Big Data Systems spectacles the demand for continuous enhancement of supporting algorithms. Hence, this article demonstrates an enhanced Bloom Filter that integrates immensely efficient programming techniques that significantly reduce the memory space used by the state-of-the-art. Besides, this new work introduces an efficient way of implementing algorithms to minimize processing time, preserving the lowest FPP.

4 Proposed System

The flowchart of our proposed work on an intrusion detection system is depicted in Figure 3. Our proposed system uses two Bloom Filters, namely, iBF and bBF. The iBF is used to store the information of intrusion data packets, and bBF is used to store benign data packet information. An incoming data packet is checked for membership in iBF. If the data packet is a member of iBF, then it will be checked for bBF. If bBF returns false then it is an intrusion. If bBF return true, it may be a case of false positive; therefore, it is tested by a deep learning method [33]. The data packet is classified as benign or intrusion. Besides, according to its classification, the data packet is inserted into either iBF or bBF. If the incoming data packet is not a member of iBF, then check the data packet for membership in bBF. If bBF returns true, then it is benign. Otherwise, the data packet is tested by the deep learning model for classification. If the data packet is an intrusion or malicious declared by the deep learning model, then insert the data packet information in iBF; otherwise, insert it in bBF. This process ensures that the already learned data packet will not be tested by the deep learning model again. It saves time and space for the deep learning model.

Training a deep learning model requires enormous computation resources such as GPU; however, testing is fast and does not require many computational resources. A trained deep learning model can be deployed in IoT devices for intrusion detection. Therefore, it requires additional data structures to reduce memory consumption and faster lookup performance. Consequently, we developed a Bloom Filter with a deep learning model that can be deployed in IoT devices. The Bloom Filter can respond to the queries already learned. Thus, it reduces the unnecessary loads on the deep learning model. For instance, BigTable uses Bloom Filter to avoid unnecessary HDD accesses [34].

For intrusion detection, we propose a novel Bloom Filter, an enhanced Bloom Filter (eBF), which is an exceedingly memory-efficient two-dimensional Bloom Filter compared with the Standard Bloom Filter. Our key focus is on designing an efficient Bloom Filter that provides a faster query response time using a small memory footprint without sacrificing accuracy. Our proposed Bloom Filter is relatively faster and at least equally accurate as robustBF [15]. Like robustBF, this proposed system uses the dimensions of the filter to speed up the insertion and searching process without degrading accuracy. This new system uses more number of memory units than robustBF. However, as Figure 4 shows, this enhanced approach employs 32-bit length for every representation unit, unlike robustBF, which uses 64-bit length. So, as the memory unit size used for every representation is half of robustBF's memory unit size, eBF reduces the memory space consumption to 8X smaller than robustBF.

eBF implements more efficient algorithms to boost the speed of insertion and searching. The time efficiency advantage of this proposed system is significantly visible when checking the existence of disjoint datasets. Like robustBF, eBF implements three hash functions for insertion and differently implements the second and third murmur hashes for searching, only if the first hash result exists in the filter. This approach minimizes the time of searching for nonexisting elements. Accordingly, eBF is more potent than Standard Bloom Filter, Counting Bloom Filter, and robustBF. The following subsections describe the components of the proposed new system.

4.1 Operations

Algorithm 1 initializes all requirements of the Bloom Filter *(BF)*. This algorithm accepts the total number of elements *(N)* to insert with the expected maximum

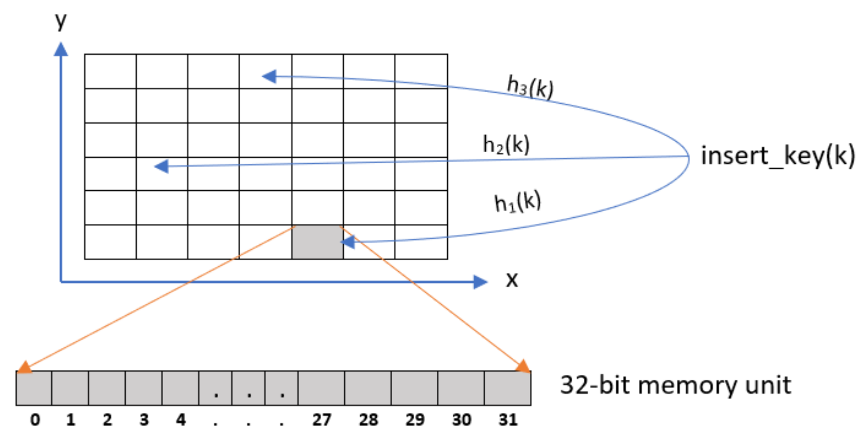


Figure 4 2DBF [14] Memory Architecture of eBF With X*Y units of 32-bit

rate of *FPP*. Then it determines the size of memory space *(M)* by calculating the maximum *(X)* and *(Y)* coordinates of the 2DBF memory structure. Consequently, it allocates the required free space for the Bloom Filter.

Algorithm 1 Initialization of Bloom Filter

```
1: procedure INIT($N$, $FPP$)
2:     Compute $M$
3:     Determine $X$ and $Y$ dimensions
4:     Allocate $X \times Y$ memory space
5:     Initialize $BF$ content to zero          ▷ Every cell is set to 0
6:     return $BF$
7: end procedure
```

Algorithm 2 inserts the three key representations to every corresponding memory location in the Bloom Filter. Every hash function sets a separate digest, and the refined result of digest against the X and Y coordinates defines the unique memory location of the key to avoid a collision. The integer value stored in the memory is a module of the cell length. Accordingly, this algorithm receives the memory allocated to the Bloom Filter, the key k, and predefined computing seeds. Thus, it passes the key and seeds to every hash function to produce different digests d. Algorithm 3

Algorithm 2 Insertion of key representation

```
1: procedure INSERT-KEY($BF, k, seed$)
2:     $i \leftarrow$ MURMUR2($k, length(k), seed$) mod $X$
3:     $j \leftarrow$ MURMUR2($k, length(k), seed$) mod $Y$
4:     $d \leftarrow$ MURMUR2($k, length(k), seed$) mod 31          ▷ Adjusting data size
5:     $BF[i][j] \leftarrow BF[i][j] | (1U << d)$          ▷ Inserting key representation
6: end procedure
```

checks the representation of the key in all three corresponding memory units. The algorithm requires element key k. It uses murmur2 hash functions to generate X and Y coordinates in a similar way the insertion is implemented. In this algorithm, the second *h2* and third *h3* hash functions are called if and only if the first hash function *h1* confirms the existence of representation for the key. If all three hash results confirm the representation of the key, the algorithm returns true, confirming the probabilistic existence of the key.

Algorithm 3 Lookup for membership of an element

```
1: procedure LOOK-UP(($BFk$)
2:     $h1 \leftarrow$ MURMUR2($k, length(k), seed1$)
3:     if TEST($BF,\ h1$) then
4:         $h2 \leftarrow$ MURMUR2($k, length(k), seed2$)
5:         if TEST($BF,\ h2$) then
6:             $h3 \leftarrow$ MURMUR2($k, length(k), seed3$)
7:             if TEST($BF,\ h3$) then
8:                 return $True$
9:             else
10:                return $False$
11:            end if
12:        else
13:            return $False$
14:        end if
15:    else
16:        return $False$
17:    end if
18: end procedure
```

5 Experimental Result

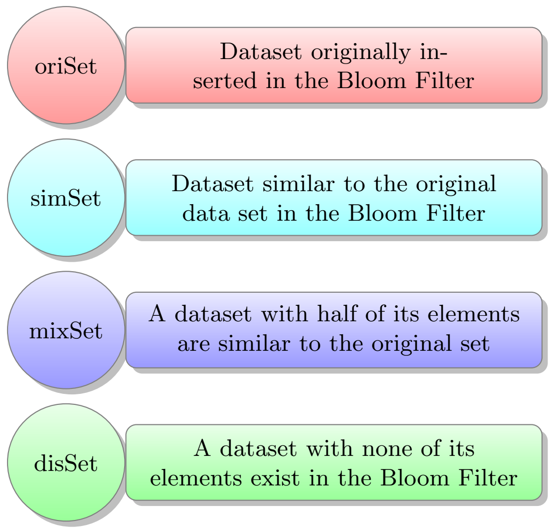
This section demonstrates the proposed system's efficiency and accuracy compared to Standard Bloom Filter, Cuckoo Filter, and robustBF. This section demonstrates the proposed system's efficiency and accuracy compared to Standard Bloom Filter, Cuckoo Filter, and robustBF. The system used to test eBF consists of a processor with the specification of Intel® Core™ i5-8250U CPU @ 1.60GHz $\times$ 8, a memory of size 8GB, and a 1TB hard disk. The operating system is a 64-bit Ubuntu 22.04 LTS.

5.1 Dataset Description

This system uses both actual data saved in CSV file format and synthetic data to evaluate the accurate rate of false positive occurrence using variant datasets across the different filtering systems. Though we have considered the FPP tested in the uncontrolled real dataset for selecting the number of hash functions, the accuracy assessment is accurate using the synthetic data. The reason to use a synthetic dataset for accuracy assessment is that the data is known to conclude on the similarity or unlikeness.

5.1.1 Synthetic Datasets

Synthetic datasets are collections of integers generated in a way they can use for the accurate evaluation of the system performance. The experimentation uses three types of datasets to test the accuracy and efficiency of the proposed filter by searching the existence of the elements in the original Set *(oriSet)*, for example, O=$\{o_1, o_2, o_3, ..., o_n\}$.


oriSet
Dataset originally inserted in the Bloom Filter
simSet
Dataset similar to the original data set in the Bloom Filter
mixSet
A dataset with half of its elements are similar to the original set
disSet
A dataset with none of its elements exist in the Bloom Filter

The first testing dataset is a set of elements similar to those represented in the Bloom Filter. So, it is known as **simSet**, for example, S=$\{o_1, o_2, o_3, ..., o_n\}$. The second testing dataset is known as **mixSet**, for example, M=$\{o_1, o_2, o_3 ... d_1, d_2, ..., d_n\}$. Its half content is intentionally changed to differ from the original dataset inserted in the Bloom Filter. **disSet** is the third dataset which consists completely different elements from the original set (O), for example, D=$\{d_1, d_2, d_3, ..., d_n\}$. The efficiency advantage of the new system increases highly in searching **disSet**.

5.1.2 Real Datasets

The real datasets used to assess the performance of the proposed scheme are available in a public repository. These real datasets are related to IoT detected intrusions from different systems at different times. An experiment that uses the actual datasets increases the feasibility of the proposed system on the IoT and similar Big Data Systems. Table 1 shows the real datasets used in this experimentation.

Table 1 Details of Real Datasets

Dataset	**Dataset Description**	**Size**	**Number of records**
DSet1	Refined IoT dataset for intrusion detection systems Without duplication. [35]	143 MB	907996
DSet2	A selected dataset for evaluating deep learning-based intrusion detection systems. [36]	1.13 GB	2219202
DSet3	IoT dataset for intrusion detection systems. [37]	1.56 GB	7062607

The datasets DSet1 and DSet3 contain network traffic sniffed from nine IoT devices using Wireshark in a local network using a central switch. It includes two Botnet attacks: Mirai and Gafgyt. The datasets contain 23 statistically engineered features extracted from the .pcap files. Seven statistical measures (variance, mean, magnitude, count, covariance, radius, correlation coefficient) have been computed. The dataset DSet3 contains data generated from more than ten types of IoT devices, i.e., Ultrasonic sensors, Low-cost digital sensors for sensing temperature and humidity, Water level detection sensors, etc. The data is related to attacks of connectivity protocol and categorized into five threats, including injection attacks, DoS/DDoS

Table 2 Comparison of insertion speed (in seconds) and false positive probability based on the number of hash functions in eBF using synthetic dataset. 10, 50, and 100 Million is the number of elements inserted into eBF.

Number	10 Million		50 Million		100 Million	
of K	Speed	FPP	Speed	FPP	Speed	FPP
1	1	0	5.7	0.00016	10.5	0.0001
2	1.35	0	6.27	0	13.5	0
3	1.7	0	8.65	0	17.35	0
4	2.15	0	11	0	22	0
5	2.37	0	12	0	23	0
6	2.48	0	13	0	25.8	0
7	2.8	0	14.6	0	29.3	0
8	3.1	0	16.3	0	32.9	0
9	3.5	0	18	0	36.6	0
10	3.9	0	19.9	0	40.7	0

attacks, man-in-the-middle attacks, information gathering, and malware attacks [36].

The experiment uses a single and more relevant column of every dataset as a key of representation. Hence, the unique key columns, i.e., "ID" of DSet1, "Triggering time" of DSet2, and values of column "variance" from DSet3, are used to represent the elements of the datasets.

5.2 Hash Function Selection

The number and type of hash function implementation significantly impact a Bloom Filter's efficiency. Several hash functions are available but murmur hash [38] is an efficient and effective non-cryptographic hash function [15, 11]. So, eBF prefers to use the murmur hash function to achieve the goals of ensuring remarkably high time and space efficiency as well as optimal accuracy. The increase in the number of hash functions increases the accuracy by minimizing false positives. However, it reduces the insertion and lookup efficiency. On the other hand, speed increases by reducing the number of hash functions but increases the FPP or requires more memory space to maintain the accuracy. So, the determination of the number of hash functions demands appropriate evaluation. Accordingly, an experiment of this new system under a fixed memory space but with a variety value of K for the synthetic data shows the result in Table 2. According to the experimental result, a single hash function (K=*1*) has the fastest speed for inserting data. However, (K=*1*) generates false positive. Using two hash functions (K=*2*) can achieve the second-fastest time but records zero false positives. Thus, implementing (K=*2*) provides optimal accuracy with good time efficiency using a synthetic dataset. Nevertheless, the evaluation of synthetic data alone cannot lead to a conclusion. So real dataset evaluation decides the number of K for better accuracy.

The efficiency and accuracy evaluation of a different number of hash functions on the real dataset is important in deciding the number of hash functions. So, the experimental result on real data shows that using two hash functions (K=*2*) recorded *FPP= 0.002* . But using three hash functions (K=*3*) has the maximum result *FPP=0.0006*. As a result, this eBF implements three hash functions (K=*3*) to achieve the best performance with at most *FPP = 0.001*.

5.3 Experiment of Synthetic Datasets

This subsection demonstrates the experimental result of eBF compared with Standard Bloom Filter, Cuckoo Filter, and robustBF using synthetic datasets. Controlled datasets effectively evaluate the filter's accuracy because the similarity or dissimilarity of two or more datasets can be accurately known only when the dataset elements are under the control of system testers. Therefore, the synthetic datasets have 10 Million (10M), 50 Million (50M), and 100 Million (100 M) records.

5.3.1 Memory Space Consumption result

The experiment result shows that eBF consumes 15x, 13x, and 8x less memory size when compared with the Standard Bloom Filter, Cuckoo Filter, and robustBF, respectively. Figure 5 clearly depicts the advantage of using eBF over the state-of-art to improve the efficiency of intrusion detection systems in IoT and query processing in Big Data Systems.

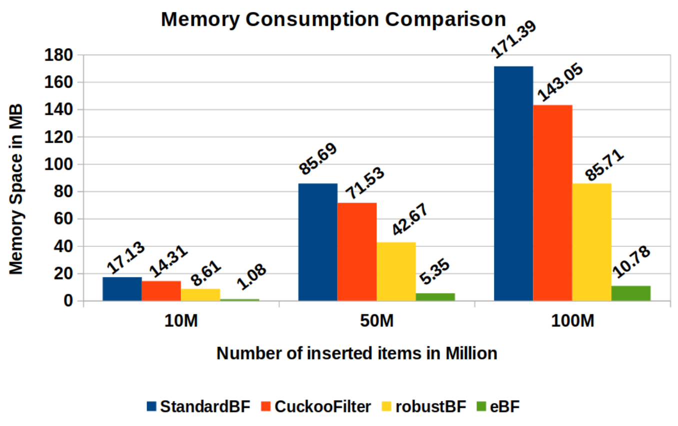


Figure 5 Memory space usage comparison among the state-of-the-art membership filters using a synthetic dataset of 10 million, 50 million, and 100 million records. The lowest memory size consumption is the best.

5.3.2 Insertion speed Comparison

Though the gap is not much visible as the memory space advantage, on average, eBF is faster than all membership filters under the same experimentation at the time of insertion and searching. Figure 6 shows how the proposed system is faster than Standard Bloom Filter, Cuckoo Filter, and robustBF.

5.3.3 Lookup speed Comparison

Searching and accuracy comparisons use synthetic datasets. These generated synthetic datasets are relevant to accurately distinguish the speed difference in searching operation and precisely show the FPP in every system under the comparison domain. Figure 7 displays the speed comparison of searching similar datasets in the Bloom Filter. In our algorithm, searching members of similar datasets demand more time than searching members of the different dataset.

The searching speed for datasets that contain members different from members of the original dataset is faster than searching similar datasets. This difference results from an algorithm we modified to ignore the process of the following hash function when the preceding hash function returns false, denying the existence of the element. For instance, taking the dataset with 100M and the second fastest system

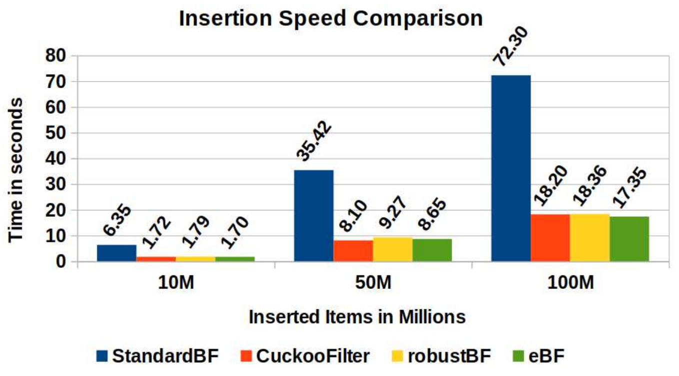


Figure 6 Insertion time comparison among the-state of-the-art membership filters synthetic dataset of 10 Million, 50 Million, and 100 Million records. The lowest is the best.

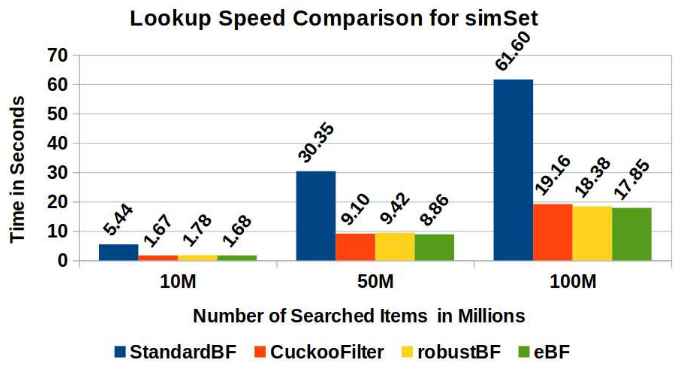


Figure 7 Searching time comparison among Standard Boom Filter, Cuckoo Filter, robustBF, and the new system eBF using a similar synthetic dataset to the dataset in the filter. The lowest time is the best

robustBF as comparing parameters, eBF uses 97% of the time used by robustBF while searching **simSet**. However, eBF only requires 87% and 76% of the time used by robustBF for searching **mixSet** and **disSet**, respectively. Figure 8 displays the searching speed comparison using a half similar and a half disjoint dataset with the dataset represented in the Bloom Filter. Figure 9 depicts how the speed of disjoint dataset searching in eBF is the fastest of all queries of the same dataset in Standard Bloom Filter, Cuckoo Filter, and robustBF. Hence, eBF is not only exceptionally memory efficient but also more time-efficient when compared with the state-of-art membership filters.

5.3.4 Accuracy assessment

Accuracy assessment evaluates the correctness of the system to deliver a proper reply to users when searching for the existence of an element in the system. Though the Bloom Filter is an efficient data structure for membership searching, it faces the challenge of providing false positives. Accordingly, identifying the probability of false positives is the main aim of this experiment.

Based on the result of the experiment, the proposed system is highly accurate. Our experiment shows that both eBF and robustBF record zero false positives and

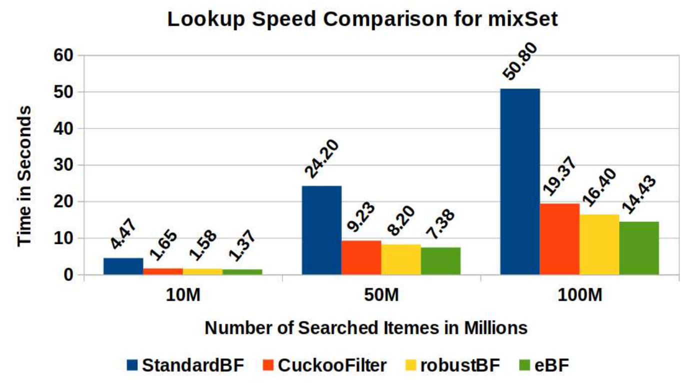


Figure 8 A search speed comparison among filters on a dataset that proportionally consists of both similar and different elements compared with the elements of the original dataset represented in the Bloom Filter. The lowest time is the best

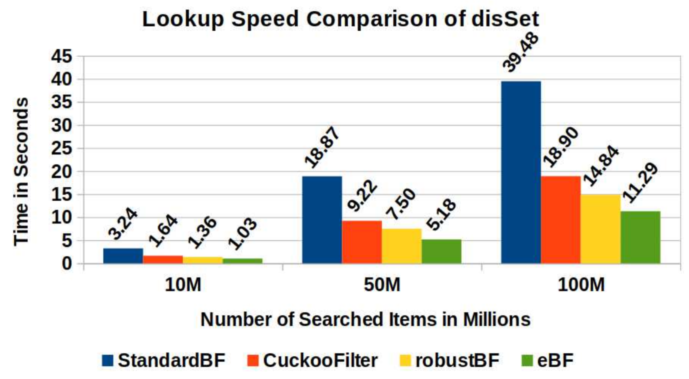


Figure 9 Searching time comparison among the filters using disjoint dataset. The lowest time is the best

zero false negatives. So it is possible to conclude that the result approves the 100% accuracy of both eBF and robustBF under synthetic datasets. However, Standard Bloom Filter and Cuckoo Filter show small false positives but no false negatives. The accuracy is calculated in terms of the ratio of the sum of True Positive (TP) and True Negative (TN) to the expected true result and the sum of False Positive occurrence FP and False Negative FN from the result of the system's output [39]. All the systems tested in this paper record zero FN. According to equation 4, Figure 10 witnesses the result of accuracy assessment based on the test of the three different synthetic datasets.

$$Accuracy = \frac{TP + TN}{(TP + TN + FP + FN)} \tag{4}$$

Hence, it is possible to conclude that eBF is a highly efficient and accurate Bloom Filter to handle Big Data membership search.

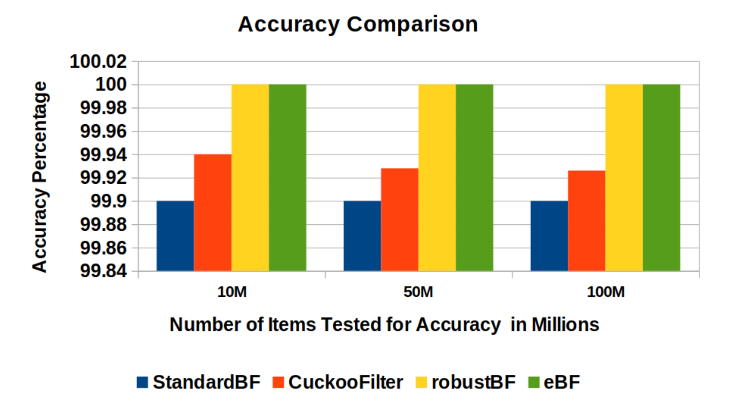


Figure 10 Accuracy comparison between robustBF and eBF. Higher is better.

5.4 Experimentation Using Real Dataset

In addition to the synthetic datasets, this new system uses real datasets accessed from an open repository. These datasets contain Intrusion Detection results of IoT systems.

5.4.1 Memory Space Comparison

Space efficiency is one of the significant features that make Bloom Filter among the essential performance enhancement tools of Big Data Systems. However, this new system eminently diminishes the memory size required to store the representation of tens and hundreds of millions of elements. eBF is 15x, 13x, and 8x memory efficient compared to the Standard Bloom Filter, Cuckoo Filter, and robustBF, respectively. Figure 11 depicts that eBF consumes the smallest memory space with a big gap against the other systems. Thus, eBF is an appropriate solution for enhancing the efficiency of Big Data processing for membership identification.

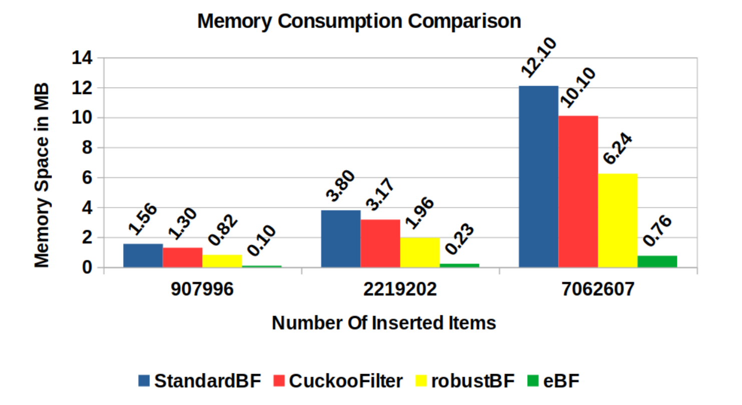


Figure 11 Memory consumption comparison between eBF and the state-of-the-art membership filters. Real datasets are used in this comparison evaluation. The lowest is the best.

5.4.2 Speed Comparison

The proposed system is more time efficient than Standard Bloom Filter, Cuckoo filter, and robustBF. Figure 12 shows how the proposed system is faster than the other systems. The efficiency gap increases when the number of elements represented in the filter increases.

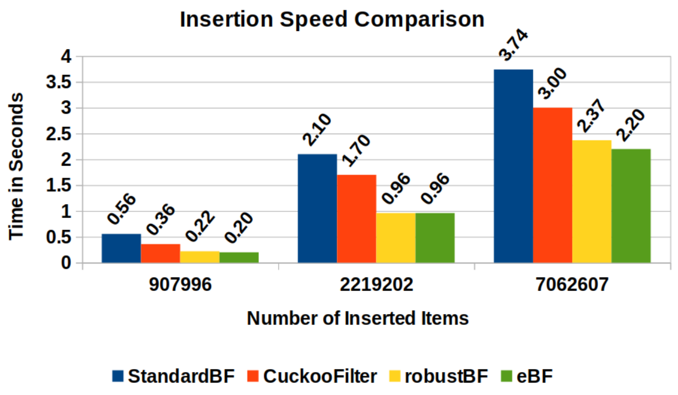


Figure 12 Real data insertion speed comparison between eBF and Standard Bloom Filter, Cuckoo Filter, and robustBF. The lowestThe lowest is the best

Figure 13 demonstrates how the frequently processed Bloom Filter's operation -lookup- is accelerated. Though fast searching is the distinguishing nature of a Standard Bloom Filter, Figure 13 shows that searching in eBF is more efficient than the state-of-the-art. Accordingly, eBF has become a necessary solution to optimize the performance of Big Data Systems.

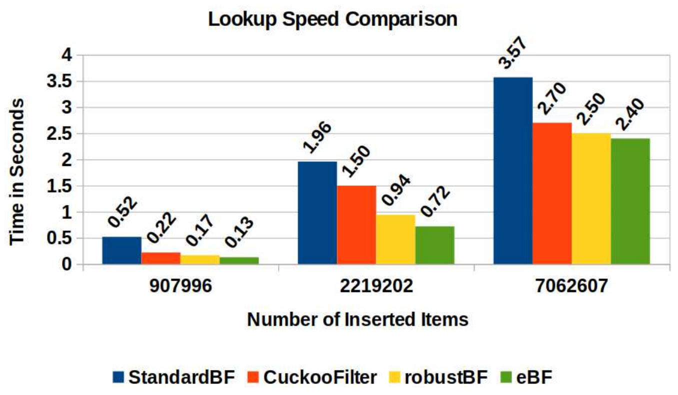


Figure 13 Real data lookup speed comparison among eBF, Standard Bloom Filter, Cuckoo Filter, and robustBF. The lowest is the best

5.4.3 Accuracy Assessment

As shown in Table 1 the real datasets used for time and space assessment are accessed from different environments. However, they may contain similar elements. So, assessing the exact rate of false positives is difficult. However, it is possible to test the false negative status by searching the same dataset over the Bloom Filter that represents it. Accordingly, the experiment shows that the Standard Bloom Filter, Cuckoo Filter, robustBF, and eBF record zero false negatives.

6 Discussion

IoT and various Big Data Systems handle an immense interaction among billions of users simultaneously. The intrusion detection systems in IoT monitor malicious events based on a predefined set of threats to prevent billions of devices from being attacked. This complex interaction demands advanced technologies that enhance computational performance [10]. Accordingly, eBF is an appropriate solution to support the robust computation of IoT intrusion detection. Moreover, it enhances the performance of complex networks of social media such as Facebook, YouTube, and WhatsApp and cloud vendors such as Microsoft, Amazon, and Google that handle the processing of Big Data. eBF surprisingly diminishes the memory space requirement of Standard Bloom Filter, Cuckoo Filter, and robustBF 15x, 13x, and 8x respectively. Key insertion to and search processing on traditional Bloom Filter requires enhancement to cope with the rapid growth of applications. eBF is faster than the Standard Bloom Filter, Cuckoo Filter, and robustBF. eBF is the best solution to support the performance enhancement of Big Data Systems based on the insertion and search speed. Moreover, eBF shows better accuracy than all the systems tested on the same data and environment. Thus, this new system significantly addresses the efficiency and accuracy issues of intrusion detection in IoT. In the future, the authors of this article plan to integrate this enhanced system with efficient and powerful cryptographic techniques to address the data security issues of Big Data Systems.

7 Conclusion

IoT integrates more than 11.3 billion devices ranging from small household appliances to sophisticated biotechnologies and industries. This broad domain range exposes the IoT network to viruses, attacks, and malicious events. So, Intrusion detection is applied to detect these threats based on priory defined security policy. Enhanced Bloom Filter is an appropriate data structure to efficiently and effectively detect incidents of attack in IoT. Besides, Big Data warehouses store data measured in petabytes (millions of gigabytes). So they need high-capacity memory or memory usability enhancement technologies. Moreover, popular social media like Facebook and YouTube have the same demand to enhance their computational capacity to satisfy billions of users with fast and reliable service. Accordingly, the proposed system presents an outstandingly high-performance data structure for Big Data processing. This hugely notable result was experimented with using big datasets with up to 100 million records. The result shows that eBF is remarkably memory efficient than the classical Bloom Filter, Cuckoo Filter, and robustBF. This system is also faster in inserting and searching operations than these three comparing systems. The speed advantage of this new system increases more when there is searching of disjoint elements from the dataset stored initially. So, the frequent search for non-existing elements cannot affect the system's performance. It also records zero false positives and zero false negatives under the properly synthetic clean datasets. This result shows that eBF is a 100% accurate Bloom Filter. The proposed system has successfully achieved all its objectives by delivering notably exceptional performance and reliability enhancements for intrusion detection systems.

Funding
There is no funding information to disclose.

Availability of data and materials
The datasets used to test the performance of this system are available at
1 https://www.kaggle.com/azalhowaide/iot-dataset-for-intrusion-detection-systems-ids?select=BoTNeTIoT-L01-v2.csv
2 https://www.kaggle.com/azalhowaide/iot-dataset-for-intrusion-detection-systems-ids?select=BotNeTIoT-L01_label_NoDuplicates.csv
3 https://www.kaggle.com/datasets/mohamedamineferrag/edgeiiotset-cyber-security-dataset-of-iot-iiot

Ethics approval and consent to participate
Not Applicable

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
All authors equally contributed.

Author details
[1]School of Computing, Mekelle University, Mekelle, Tigray-Ethiopia. [2]Department of Computer Science and Engineering, National Institute of Technology, Silchar, 788010 India.

References
1. Tewari, A., Gupta, B.B.: Security, privacy and trust of different layers in internet-of-things (iots) framework. Future Generation Computer Systems **108**, 909–920 (2020). doi:10.1016/j.future.2018.04.027
2. Yadav, K., Gupta, B.B., Hsu, C.-H., Chui, K.T.: Unsupervised federated learning based iot intrusion detection. In: 2021 IEEE 10th Global Conference on Consumer Electronics (GCCE), pp. 298–301 (2021). doi:10.1109/GCCE53005.2021.9621784
3. Adel, A.: Utilizing technologies of fog computing in educational iot systems: privacy, security, and agility perspective. Journal of Big Data **7**(1), 1–29 (2020). doi:10.1186/s40537-020-00372-z
4. Vailshery, L.S.: Number of iot connected devices worldwide 2019-2030 (Accessed: July, 2022)
5. Zuech, R., Khoshgoftaar, T.M., Wald, R.: Intrusion detection and big heterogeneous data: a survey. Journal of Big Data **2**(1), 1–41 (2015). doi:10.1186/s40537-015-0013-4
6. Honar Pajooh, H., Rashid, M.A., Alam, F., Demidenko, S.: Iot big data provenance scheme using blockchain on hadoop ecosystem. Journal of Big Data **8**(1), 1–26 (2021). doi:10.1186/s40537-021-00505-y
7. Putra, G.D., Dedeoglu, V., Kanhere, S.S., Jurdak, R.: Poster abstract: Towards scalable and trustworthy decentralized collaborative intrusion detection system for iot. In: 2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI), pp. 256–257 (2020). doi:10.1109/IoTDI49375.2020.00035
8. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970). doi:10.1145/362686.362692
9. Head, P.: Area efficient counting bloom filter (a-cbf) design for nids. International Journal of Computer Applications **975**, 8887
10. Mosharraf, S.I.M., Adnan, M.A.: Improving lookup and query execution performance in distributed big data systems using cuckoo filter. Journal of Big Data **9**(1), 1–30 (2022). doi:10.1186/s40537-022-00563-w
11. Patgiri, R., Nayak, S., Muppalaneni, N.B.: Is bloom filter a bad choice for security and privacy? In: 2021 International Conference on Information Networking (ICOIN), pp. 648–653 (2021). doi:10.1109/ICOIN50884.2021.9333950
12. Patgiri, R., Nayak, S., Borgohain, S.K.: Role of bloom filter in big data research: A survey. arXiv preprint arXiv:1903.06565 (2019)
13. Nayak, S., Patgiri, R.: countbf: A general-purpose high accuracy and space efficient counting bloom filter. In: 2021 17th International Conference on Network and Service Management (CNSM), pp. 355–359 (2021). doi:10.23919/CNSM52442.2021.9615556
14. Patgiri, R., Nayak, S., Borgohain, S.K.: rdbf: A r-dimensional bloom filter for massive scale membership query. Journal of Network and Computer Applications **136**, 100–113 (2019)
15. Nayak, S., Patgiri, R.: Robustbf: A high accuracy and memory efficient 2d bloom filter. arXiv preprint arXiv:2106.04365 (2021)
16. Fan, B., Andersen, D.G., Kaminsky, M., Mitzenmacher, M.D.: Cuckoo filter: Practically better than bloom. In: Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies. CoNEXT '14, pp. 75–88. Association for Computing Machinery, New York, NY, USA (2014). doi:10.1145/2674005.2674994
17. Guo, D., Liu, Y., Li, X., Yang, P.: False negative problem of counting bloom filter. IEEE Transactions on Knowledge and Data Engineering **22**(5), 651–664 (2010). doi:10.1109/TKDE.2009.209
18. Patgiri, R.: Hfil: A high accuracy bloom filter. In: 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 2169–2174 (2019). doi:10.1109/HPCC/SmartCity/DSS.2019.00300. IEEE
19. Kiss, S.Z., Hosszu, E., Tapolcai, J., Ronyai, L., Rottenstreich, O.: Bloom filter with a false positive free zone. IEEE Transactions on Network and Service Management **18**(2), 2334–2349 (2021). doi:10.1109/TNSM.2021.3059075

20. Gerbet, T., Kumar, A., Lauradoux, C.: The power of evil choices in bloom filters. In: 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 101–112 (2015). doi:10.1109/DSN.2015.21
21. Patgiri, R., Nayak, S., Muppalaneni, N.B.: Is bloom filter a bad choice for security and privacy? In: 2021 International Conference on Information Networking (ICOIN), pp. 648–653 (2021). doi:10.1109/ICOIN50884.2021.9333950
22. Artan, N.S., Sinkar, K., Patel, J., Chao, H.J.: Aggregated bloom filters for intrusion detection and prevention hardware. In: IEEE GLOBECOM 2007 - IEEE Global Telecommunications Conference, pp. 349–354 (2007). doi:10.1109/GLOCOM.2007.72
23. Groza, B., Murvay, P.-S.: Efficient intrusion detection with bloom filtering in controller area networks. IEEE Transactions on Information Forensics and Security **14**(4), 1037–1051 (2019). doi:10.1109/TIFS.2018.2869351
24. Bala, P.M., Usharani, S., Aswin, M.: Ids based fake content detection on social network using bloom filtering. In: 2020 International Conference on System, Computation, Automation and Networking (ICSCAN), pp. 1–6 (2020). doi:10.1109/ICSCAN49426.2020.9262360
25. Zinkus, M., Khosmood, F., DeBruhl, B.: Pidiot: Probabilistic intrusion detection for the internet-of-things. In: 2019 IEEE Global Communications Conference (GLOBECOM), pp. 1–6 (2019). doi:10.1109/GLOBECOM38437.2019.9013264
26. Harwayne-Gidansky, J., Stefan, D., Dalal, I.: Fpga-based soc for real-time network intrusion detection using counting bloom filters. In: IEEE Southeastcon 2009, pp. 452–458 (2009). doi:10.1109/SECON.2009.5174096
27. Todorov Marinov, M.: A bloom filter application for processing big datasets through mapreduce framework. In: 2021 International Conference on Information Technologies (InfoTech), pp. 1–5 (2021). doi:10.1109/InfoTech52438.2021.9548638
28. Singh, A., Garg, S., Kaur, R., Batra, S., Kumar, N., Zomaya, A.Y.: Probabilistic data structures for big data analytics: A comprehensive review. Knowledge-Based Systems **188**, 104987 (2020). doi:10.1016/j.knosys.2019.104987
29. Kiss, S.Z., Hosszu, E., Tapolcai, J., Ronyai, L., Rottenstreich, O.: Bloom filter with a false positive free zone. IEEE Transactions on Network and Service Management **18**(2), 2334–2349 (2021). doi:10.1109/TNSM.2021.3059075
30. Harshan, J., Vithalkar, A., Jhunjhunwala, N., Kabra, M., Manav, P., Hu, Y.-C.: Bloom filter based low-latency provenance embedding schemes in wireless networks. In: 2020 IEEE Wireless Communications and Networking Conference (WCNC), pp. 1–7 (2020). doi:10.1109/WCNC45663.2020.9120640
31. Lucchesi, A., Drummond, A.C., Teodoro, G.: High-performance ip lookup using intel xeon phi: a bloom filters based approach. Journal of Internet Services and Applications **9**(1), 1–18 (2018)
32. Fan, L., Cao, P., Almeida, J., Broder, A.Z.: Summary cache: a scalable wide-area web cache sharing protocol. IEEE/ACM Transactions on Networking **8**(3), 281–293 (2000). doi:10.1109/90.851975
33. Vinayakumar, R., Alazab, M., Soman, K.P., Poornachandran, P., Al-Nemrat, A., Venkatraman, S.: Deep learning approach for intelligent intrusion detection system. IEEE Access **7**, 41525–41550 (2019). doi:10.1109/ACCESS.2019.2895334
34. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A distributed storage system for structured data. ACM Trans. Comput. Syst. **26**(2) (2008). doi:10.1145/1365815.1365816
35. Kaggle's Non Duplicated IoT Dataset for Intrusion Detection Systems (IDS). `https://www.kaggle.com/azalhowaide/iot-dataset-for-intrusion-detection-systems-ids?select=BotNeTIoT-L01_label_NoDuplicates.csv`
36. Edge-IIoTset Cyber Security Dataset. `https://www.kaggle.com/datasets/mohamedamineferrag/edgeiiotset-cyber-security-dataset-of-iot-iiot`
37. Kaggle's IoT Dataset for Intrusion Detection Systems (IDS) With Duplication. `https://www.kaggle.com/azalhowaide/iot-dataset-for-intrusion-detection-systems-ids?select=BoTNeTIoT-L01-v2.csv`
38. Austin, A.: Murmurhash (Accessed: June, 2022)
39. Tharwat, A.: Classification assessment methods. Applied Computing and Informatics (2020). doi:10.1016/j.aci.2018.08.003