Received: November 2, 2020. Revised: November 26, 2020.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Scalable Two-Dimensional Bloom Filter Membership Scheme on Massive Scale Wireless Sensor Networks

Farah Afianti[1]* **Cahya Asrini[1]** **Dedy Rahman Wijaya[2]**

[1]*School of Computing, Telkom University, Bandung 40257, Indonesia*
[2]*School of Applied Science, Telkom University, Bandung 40257, Indonesia*
* Corresponding author's Email: farahafi@telkomuniversity.ac.id

Abstract: Wireless sensor nodes help people to monitor any operational mechanism. It is important to be implemented in industrial automation in detecting unauthorized activity or malicious sensor nodes by the outsider. However, a massive number of devices or things which are needed to be controlled become a challenge. A fast mechanism to recognize a member or a valid device is needed to minimize production costs and avoid future procedural errors. It becomes an open problem for the implementation of massive scale Wireless Sensor Networks (WSN). Hence, a scalable two-dimensional Bloom filter is proposed in this paper to deal with this issue. The low computation complexity and the storage efficiency requirements by the two-dimensional Bloom filter are proven by the experimental results. The insertion, deletion and query procedure of the proposed method only need O (1), while the storage usage of two-dimensional Bloom filters not only lower than counting Bloom filter in average 131 bits difference but also approaches accommodative Bloom filter in about 10 bits difference. These two parameters support a fast membership scheme. Furthermore, the analysis of value initialization has been performed. To the best of our knowledge, the investigation of this parameter has not been addressed by other studies. This process aims to get the best scenario to increase scalability. Several recommendations for selecting dimension number has been stated which is useful for efficient storage usage and reduce false positive.

Keywords: Dimensional bloom filter, Massive-scale, Scalable, Wireless sensor networks.

1. Introduction

Nowadays, devices or machines can be automatically monitored, based on sensor node information. This avoids human interaction which manually and physically detects each machine's behavior. Wireless sensor nodes aim to connect things to the system through its senses ability such as based on movement [1], gas concentration [2], moisture [3], etc. However, sensor nodes are resource-constraint devices which have limitation in the power, storage, and computation. Therefore, they need a lightweight mechanism to implement their daily computation and extend their operational lifetime.

Monitoring a machine needs several sensor nodes based on each sense. The factory has a large amount of machines to be monitored or product quality control to comply with the international standard [4], so it needs a lot of sensor nodes. The data from sensor nodes are transmitted to the base station. It can be continuously or periodically performed. Therefore, the base station needs to confirm whether the data is valid from the registered sensor node. Furthermore, several sensor nodes need to communicate with each other. They may exchange data or forward information that aims to reduce response time, network delay, latency among devices [5]. The challenge for carrying out this process is how to do it computationally cheap and time-efficient in determining valid sensor nodes.

Bloom filter is a probabilistic data structure that is suitable to provide a fast membership scheme in the wireless sensor network. It needs low computation because only uses a hash function to map a member's vector position [6]. It was developed to decrease storage on the receiver side. This method can be implemented in the detection of RFID systems for filtering from unregistered tag [7, 8], database of password systems [9], security issues [10], data science [11, 12], caching management [13], etc. However, the false positive result becomes a limitation in its implementation. There is a possibility of an unregistered member to be detected as a valid member. The Bloom filter variants are developed to optimize their functionality and reduce the false positive.

In this paper, a scalable two-dimensional Bloom filter is proposed as a membership scheme in wireless sensor networks. This method is fast and has low computation that is suitable for utilization on a huge number of wireless sensor networks. It is because this method has a simple procedure whether for querying, inserting, or deleting a member. Furthermore, there is an initialization parameter that needs to be adjusted. To the best of our knowledge, the investigation of this parameter has not been addressed by other studies. Therefore, this paper aims to analyze and optimize setting parameter values to increase its scalability. Thus, the main contribution of this study is to compare and select Bloom filter methods that are quickly characterized by low computational complexity and parameter initialization analysis to increase scalability on the massive WSN scale.

The organization of this paper is as follows: Section 2 discusses the implementation of previous methods. Section 3 explains the detail of the two-dimensional Bloom filter as proposed methods. Section 4 discusses the implementation and analysis of the result. The last section is summarized in the paper content. In addition, the table of notation is summarized in Table 1.

Table 1. Nomenclature

Symbol	Description
FPP	False Positive Probability
FPR	False Positive Rate
FPR_{ABF}	False Positive Rate of accommodative Bloom filter
C	Critical factor
α	False positive intolerance
n	The number of members
m	Size of Bloom filter
k	Number of hash function
fp	Size of fingerprint in the TinySet
B	Number of Bucket in the TinySet
UID_i	The unique identifier of ith member
PK_i	The public key of ith member
m_s	The size of Bloom filter in the second layer
N_s	The maximum accepted member in each slice of the second layer
k_1	The number of hash function in second layer which also means the number of partitions
b	The number of buckets in the first layer
l	The size of bucket in the first layer

2. Related works

Bloom filter can be grouped into five types. First is the original Bloom filter [14] which maps a member's position into a single vector. This method is very simple and has low computation complexity. It has drawbacks in the false positive although there is no possibility of a false negative. Furthermore, this method does not support member deletion. Second is counting Bloom filter (CBF) [15]. This method is the improvement of the original Bloom filter which supports member deletion. It adds several bits, usually 4-bit length, which acts as a counter. This counter is positioned in each bit of the Bloom filter vector. Third is the fingerprint-based Bloom filter. Several methods are using this idea such as TinySet [16], sliding counting Bloom filter [17], Cuckoo filter [18], etc. The advantage of those methods is low false positive because a member is not only represented by its position in the Bloom filter vector but also by its several bits of the hash value. The member position in the TinySet is placed in three parameters such as bucket (B), chain (L), and fingerprint (fp). The size of those parameters can be fixed or dynamic. There is more addition procedure which is needed for dynamic size than fixed size [16]. Furthermore, this method supports member deletion by removing fingerprints in a specific position based on the member's unique value and shifting the other fingerprint to obtain efficient storage. Fourth is the hierarchical Bloom filter. Several methods are using this idea such as hierarchical segmented Bloom filter [7], Bloom filter-based framework [19], Accommodative Bloom filter (ABF) [20], etc. This type of Bloom filter utilizes several Bloom filters at multiple levels to increase scalability. Accommodative Bloom filter has 2 layers of Bloom filter such as bucket (first layer) and each bucket has a partition of Bloom filter (second layer) [20]. The number of hash function, size of Bloom filter in the first and second layers are different. The number of hash functions in the second layer affects the number of partitions. Furthermore, there are two main computations based on mechanism in each layer. Therefore, this type of Bloom filter has a limitation in high computation in the query and addition procedure. The last type is the multidimensional Bloom filter or rDBF [21]. The idea is a modification of the hierarchical Bloom filter but in a simple manner. It consists of several dimensions of Bloom filter ranging from 2 to 5. The higher dimension of

Bloom filters the more time is needed for adding, querying, and deleting processes and the lower false positive probability [21]. This paper proposed a two-dimensional Bloom filter because of its simplicity. It is an appropriate method to be implemented in the resource constraint device such as wireless sensor networks. However, it has a limitation in scalability. Therefore, the initial setting parameter for creating the two-dimensional Bloom filter must be analyzed to avoid another cost for recreating the new membership scheme.

3. Two-dimensional bloom filter

The sensor nodes are identified by their UID (Unique Identifier) and the public key. Those values are unique for each node. The length of UID varies from 8 to 48 bits which depends on the protocol in the lower OSI layer [22]. While the public key is used for key-based authentication. This paper used the Elliptic Curve Digital Signature Algorithm (ECDSA) [23]. It is because with the same security level ECDSA only needs 40 bytes key length while RSA needs 128 bytes [24].

The proposed scheme is shown in Figure. 1. Registering and querying members in two-dimensional Bloom filters have the same procedure. The first step is concatenating UID and PK of sensor nodes. For example, in Figure. 1 UID$_3$ is concatenated with PK$_3$. Then the result is hashing using any hash function. In this paper, SHA1 is used as a cryptographic hash function with a length of 160 bits [25]. This value is converted into decimal and translated into three parameters such as first, second, and third dimension which is denoted as $x, y,$ and σ. The first and second dimension must be prime number to avoid collision [21]. Those parameters produce a cell in the form of a single Bloom filter vector with length σ bits. Based on the example in Figure. 1, the cell position is shown in the blue character, and its Bloom filter is pointed by an orange arrow. The blue character in this Bloom filter is marked by 1 which means the vector position of UID$_3$| PK$_3$ is in the 60th bit.

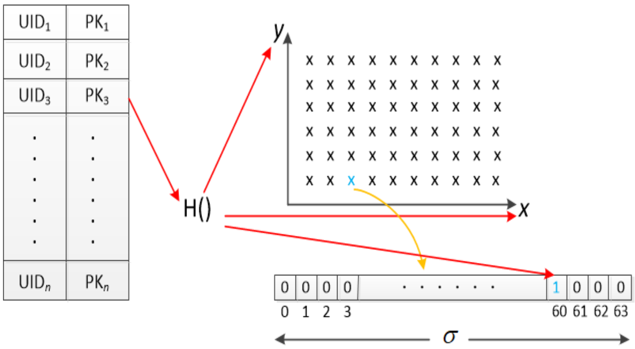

Figure. 1 Two-dimensional bloom filter on WSN

Another feature of the two-dimensional Bloom filter is member deletion. The first step is the same as registering and querying members. Once the location of the member to be removed in the Bloom filter vector is found, update the value from 1 to 0.

Bloom filter membership query has the possibility of false positive in querying procedure. This parameter can be calculated using false positive probability which is shown in Eq. (1) [21]:

$$FPP = \sum_{i=0}^{m} \left(\frac{i}{m}\right) \binom{m}{i} \sum_{j=0}^{i} (-1)^j \binom{i}{j} \left(\frac{i-j}{m}\right)^n \quad (1)$$

where,

$$m = C.x.y$$

and

$$C = \frac{\sigma}{\alpha}.$$

False positive can be also measured by Eq. (2) [26]. The number of hash functions in a two-dimensional Bloom filter is only one, so the value of k is 1. The critical factor (C) is a measurement to determine the empty space in the third dimension of 2DBF. It ranges from 1 to the maximum size of third dimension (σ). The higher value of C, the more important of false positive meaning to the 2DBF.

$$FPR = \left(1 - e^{\frac{-kn}{m}}\right)^k = 1 - e^{-\frac{n}{m}} \qquad (2)$$

According to these equations, this paper aims to adjust the best value of m by considering the low value of false positive either measured by Eq. (1) or Eq. (2).

The false positive measurement for the hierarchical Bloom filter is different. The false positive probability for each layer must be determined. The total false positive probability value is obtained from intersection among the result of false positive probability in each layer. Therefore, the false probability of accommodative Bloom filter can be measured using Eq. (3) [20]:

$$FPR_{ABF} = f_I^{ABF} . f_{II}^{ABF} \qquad (3)$$

where,

$$f_I^{ABF} = \left(1 - e^{\frac{-kn}{b}}\right)^k,$$

$$f_{II}^{ABF} = 1 - \left(\prod_{s=1}^{s=i} \left(1 - f_s^{ABF}\right)\right),$$

and

$$f_s^{ABF} = \left(\left(1 - e^{\frac{-k_1 N_S}{m_s}}\right)^{k_1}\right)^b .$$

The first layer, the second layer, and the slice in the second layer of accommodative Bloom filter are denoted as f_I^{ABF}, f_{II}^{ABF}, and f_S^{ABF}, respectively.

4. Experimental design

The detailed environment to implement the proposed method is shown in Table 2. Python programming language is chosen because of its simplicity and features, which makes the proposed method easier to be implemented.

5. Experimental scenario

There are five scenarios for each method which aim to analyze one cycle of insertion, querying, and deletion. All proposed scenarios provide space for storing the same number of members. In this experiment, four methods will be analyzed. All of them support member deletion with a different mechanism except the accommodative Bloom filter [20].

The first method is CBF and its detailed scenario can be seen in Table 3 The counter length is 4 bits. Each scenario has different size of Bloom filter and the number of hash function to analyze the false positive parameter.

The second method is TinySet [16]. There are two types of TinySet which are differentiated by its block size. TT32 stands for TinySet using 32 bits array while TT64 stands for TinySet using 64 bits array of block. The detailed scenarios of TT32 and TT64 are shown in Table 4 and Table 5, respectively.

Table 2. Experimental environment

Name	Specification
CPU	Intel® Core ™ i5-9300HF CPU @ 2.40GHz
Number of CPU	8
RAM	8192 MB
HDD	1 TB
Programming Language	Python

Table 3. CBF's scenarios

CBF	Sc1	Sc2	Sc3	Sc4	Sc5
k	1	2	3	4	5
m (bit)	130	260	390	520	650

Table 4. TT32's scenarios

TT32	Sc1	Sc2	Sc3	Sc4	Sc5
fp (bit)	4	10	16	22	28
B	3	3	3	3	3

Table 5. TT64's scenarios

TT64	Sc1	Sc2	Sc3	Sc4	Sc5
fp (bit)	2	7	11	16	20
B	2	2	2	2	2

Table 6. ABF-1's scenarios

ABF-1	Sc1	Sc2	Sc3	Sc4	Sc5
m_s	37	57	68	82	88
k_1	3	4	5	6	7

Table 7. ABF-2's scenarios

ABF-2	Sc1	Sc2	Sc3	Sc4	Sc5
m_s	45	71	84	102	109
b	6	8	10	12	14

The fourth method is the Accommodative Bloom filter (ABF) [20]. There are two types of ABF which are classified based on its purpose. ABF-1 focuses on extending the second level of Bloom filter whether by adding the partition number or the size of Bloom filter in each slice. The detailed scenario of ABF-1 is shown in .Table 6 The number of N_s, k, l and b are 20, 3, 4, and 5, respectively. It is fixed for each scenario.

ABF-2 aims to analyze the first level of Bloom filter, so it extends the value and size of bucket. Furthermore, this method extends several bits of the second layer Bloom filter to get a fair comparison in the size of accepted member in the ABF. The detailed scenario of ABF-1 is shown in Table 7. The number of N_s, l, k, and k_1 are 25, 4, 3, and 2, respectively. It is fixed for each scenario.

The last method is 2DBF which stands for two-dimensional Bloom filter. There are two types of 2DBF which are differentiated by its analysis purpose. 2DBF-1 aims to analyze the size of the Bloom filter in each cell while the size of the first and second dimensions of the cell is fixed. 2DBF-2 aims to analyze the size of the first and second dimensions of the cell while the size of the Bloom filter is fixed. The detailed scenario of 2DBF-1 is shown in Table 8. The number of x and y is 3.

The detailed scenario of 2DBF-2 is shown in Table 9. The number of C is range from 36 up to 59.

Table 8. 2DBF-1's scenarios

2DBF-1	Sc1	Sc2	Sc3	Sc4	Sc5
C	60	120	180	170	340
σ	64	130	190	280	350

Table 9. 2DBF-2's scenarios

2DBF-2	Sc1	Sc2	Sc3	Sc4	Sc5
x	3	3	3	3	3
y	5	7	11	13	17

All of the scenarios in each method are compared in the same environment. They have the same purpose to provide space for 90 members. This number is fixed and chosen to simplify the parameter comparison among CBF, TT32, TT64, ABF-1, ABF-2, 2DBF-1, and 2DBF-2.

6. Results and discussion

Aforementioned, the false positive measurement could be implemented in two ways such as FPP and FPR. The false positive in the r-dimensional Bloom filter is usually measured by using FPP [21]. However, in its implementation, there are some unstable results because of repeated multiplications required to compute the binomials and powers. It makes significant digits disappear and the inner summation with alternating signs [27]. Furthermore, the difference false positive measurement between as FPP and FPR in a simple case of 2-DBF can be seen in Figure. 2. The higher value of the first and second dimension, the lower deviation between FPP and FPR. It shows that the values of FPP and FPR in 2-DBF are close to the same.

Moreover, it is supported by the result of FPR which is shown in Figure. 3. The higher value of the first and second dimensions, the lower FPR value. It means that the difference value between FPP and FPR become insignificant. Therefore, FPR is used as parameter to compare 2DBF with the other methods in this paper.

The FPR value among several scenarios in each Bloom filter variation method is shown in Figure. 4. The value of FPR in the CBF is the highest which is occurred in all scenarios. The decreasing value of FPR cannot compete with other methods even if the number of the hash function is increased in each scenario. The values of FPR in the 2DBF-1 and 2DBF are lower than in the CBF. However, it is higher than the values of FPR in the ABF-1, ABF-2, TT32 and TT64. This value is decreased by an average 33.06% between scenarios. It occurred almost in all scenarios except in the first scenario (Sc1) of TT64. The FPR value of 2DBF-1 and 2DBF-2 is lower than TT64 by a difference of about 0.0226. However, decreasing the FPR value in other scenarios cannot compete with TT64, and certainly, TT32, although 2DBF-1 increases the critical factor in the third dimension or 2DBF-2 increases the value of the second dimension. Furthermore, the value of FPR in the 2DBF-1 and 2DBF-2 are almost the same for each scenario. It means the increasing dimension, whether by increase the second dimension or critical factor in the third dimension, does not affect the FPR. TT32 has a lower FPR value than TT64 and ABF-1 except for the first scenario (Sc1). The FPR value of ABF-1 is lower than TT32 only in the first scenario by a difference of about 0.02099. The FPR value of TT32 decreased by approximately 98.4375% between scenarios. The FPR value of TT64 is higher than TT32. The FPR value of TT64 decreased, ranging from 93.75% to 96.875% between scenarios. ABF-2 has the lowest FPR. This value is decreased by average 98.48157% between scenarios. The FPR value of ABF-1 is higher than TT64, TT32, and ABF-2 in all scenarios except Sc1. The FPR value of ABF-1 decreased by average 44.57% between scenarios.

The storage required by each method is shown in Figure. 5. CBF requires the highest storage space because it needs 4 bits counter to support member deletion for each bit Bloom filter vector. It means one member needs 5 bits which consists of 1-bit position

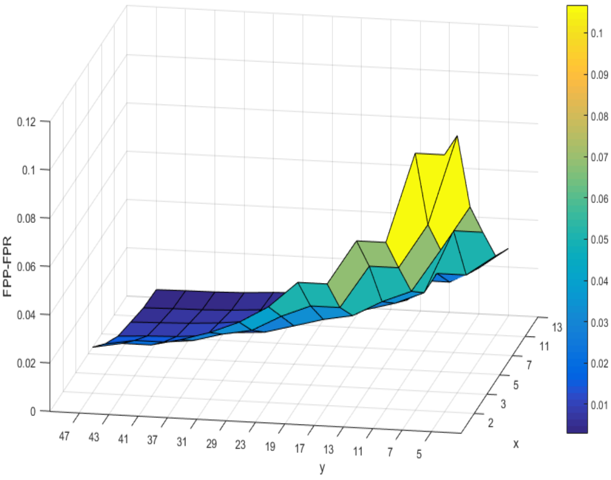

Figure. 2 Difference between FPP and FPR on two-dimensional Bloom filter using C=1 and n is 95% filled

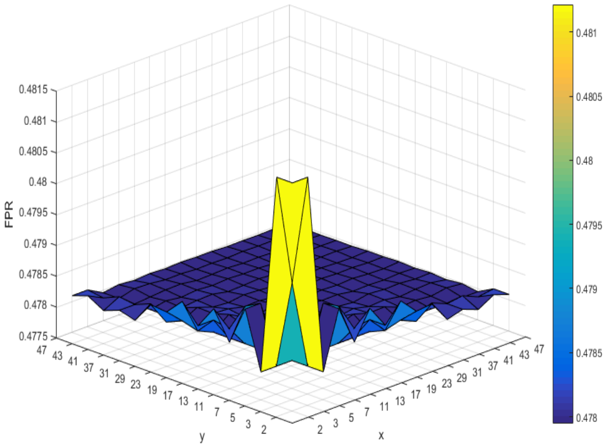

Figure. 3 FPR on 2-DBF using C=16 and n is 65% filled

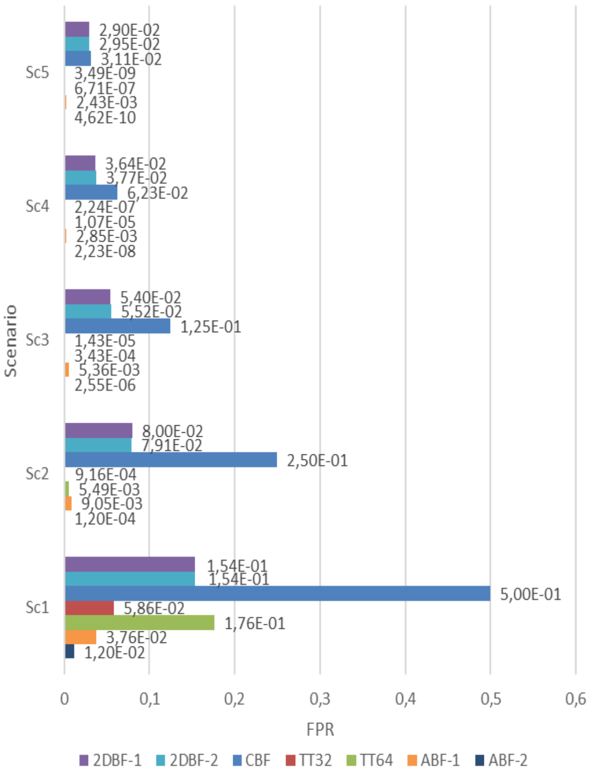

Figure. 4 FPR measurement of 2DBF-1, 2DBF-2, CBF, TT32, TT64, ABF-1, and ABF-2

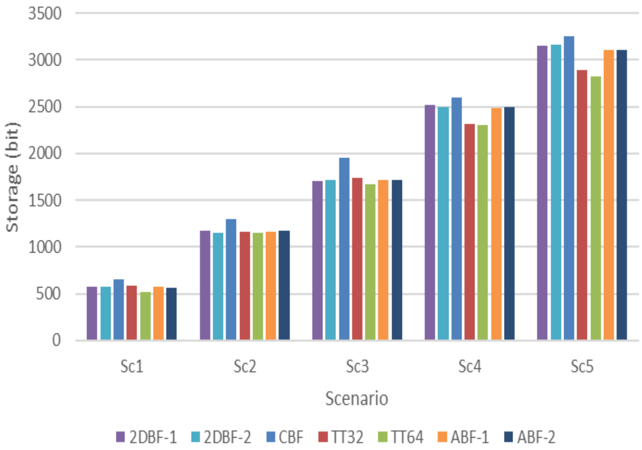

Figure. 5 Storage requirement of 2DBF-1, 2DBF-2, CBF, TT32, TT64, ABF-1, and ABF-2

and 4 bits counter. Storage usage of 2DBF-1 and 2DBF-2 was lower than CBF, with an average 131 bits difference in all scenarios. However, the storage requirement of both types in 2DBF is higher than the TT32. It is because 2DBF needs more unallocated space to increase false positive value while TT32 avoids it by using fingerprint. TT64 requires the lowest storage space because it uses a lower number of buckets and fingerprint length than TT32, even though the size of fingerprint container of TT64 is higher than TT32. ABF-1, and ABF-2 require almost the same storage space. This value is higher than TT32 dan TT64 because both types of ABF needs 2 layers of Bloom filter either for a bucket or for a slice. Each layer has its size. The total storage requirement of the second layer is the multiplication of the first layer and the size of the second layer of the Bloom filter. Therefore, the storage requirement in the ABF approaches 2DBF. The difference among them is about 10 bits. It occurred because both ABF and 2DBF have more than one Bloom filter vector representation to locate the membership position.

The computation complexity comparison among the proposed method and other methods can be seen in Table 10. The 2DBF has the lowest computation complexity because of 2 reasons. First, 2DBF needs one bit to represent a member so it only assigns zero for non-member and one for a member. Second, 2DBF only need one hash function to compute the position of first, second, and third dimension. Furthermore, the false positive value in the CBF only depends on the number of hash function (k) even a member representation only needs 1 bit. The higher value of k, the higher computation complexity. It occurs in the insertion, query, and deletion procedure. Moreover, the representation of a member in TT32 and TT64 is using a fingerprint (fp). It increases the efficiency of storage usage. The longer of fp length, the higher value of FPR. However, it increases computation complexity to put fp into specific position in the block R and chain L. The procedure to discover a chain, compare the fp, and shift the other fp is denoted by s. The lower number of Block (R) decreases the computation complexity. Therefore, the computation of s in TT32 is lower than in TT64 [10]. This paper analyses a fixed size of R and fp. The dynamic size of those two parameters increase complexity of s [16]. The ABF has the highest computation complexity because its mechanism depends on the two different values of hash function and the number of buckets. The hash computation using k and $k1$ is occurred in each bucket, so multiplication of the total number of hash functions is used in the insertion procedure. The query

Table 10. Computation comparison

Name	CBF	TT32	TT64	2DBF	ABF
Bits/ item	1	fp	fp	1	$k.k1$
Number hash	k	1	1	1	$k + k1$
Insertion	O(k)	O(s)	O(s)	O(1)	O($b(k + k1)$)
Query	O(k)	O(s)	O(s)	O(1)	O($k + k1$)
Deletion	O(k)	O(s)	O(s)	O(1)	-

Table 11. 2DBF-3's scenarios

2DBF-3	Sc1	Sc2	Sc3	Sc4	Sc5
x	5	7	5	7	7
y	7	7	11	11	13
σ	18	23	31	37	41

procedure of ABF only depends on the number of the hash function in the first and second layers because the computation in each layer can be done separately. Therefore, the computation complexity of query procedure is lower than insertion procedure in the ABF. The deletion procedure is not supported by ABF because each Bloom filter position may be owned by other members too.

This paper focused on a fast Bloom filter algorithm for a massive scale of sensor nodes. Therefore, 2DBF is the most appropriate for this case study. It has low computation complexity although has a higher false positive rate than a TinySet and ABF. It also has a lower storage requirement than CBF. Furthermore, 2DBF supports member removal with only one hash function operation.

The challenge in 2DBF is the initialization parameter to determine the value of the dimension and its Bloom filter vector or size of the third dimension. Prime number dimensions must be fulfilled to avoid collision [21]. However, there are a lot of prime numbers that can be chosen. For this purpose, another scenario has been added. The detailed parameter value for 2DBF-3 can be seen in Table 11. This version of 2DBF focuses on increasing the value in all dimensions. Moreover, each scenario in 2DBF-3 still has the same storage requirement and FPR as 2DBF-1 and 2DBF-2. It aims to make fair comparison among scenario in different version of 2DBF.

The number of members which is registered in the Bloom filter vector is analyzed. In fact, there are 90 members that are added into 2DBF but not all of them are uniquely registered because one position in the Bloom filter vector may contain more than one member. The result is shown in Figure. 6. Another parameter that supports initialization parameter analysis is the number of empty cells in 2DBF. This means for each Bloom filter vector in the cell may not contain any member. There are possibilities of uneven placement of members. This parameter result is shown in Figure. 7. The 2DBF-1 has the same value for the first and second dimensions. This has an impact on the number of empty cells which is about 66.67% of the total available cell. It also occurred in the Sc2 of 2DBF-3 which has about 85.71% empty cell. The cell dimension of Sc2 of 2DBF-3 is higher than all scenarios in the 2DBF-1. Therefore, the percentage of an empty cell in the Sc2 of 2DBF-3 is higher than 2DBF-1. The cell positions that are zero in 2DBF-1 and Sc2 in 2DBF-3 will remain empty, although the size of the Bloom filter in the third dimension is increased. It is because any member will get the same value, either for the first or second dimension. This version of 2DBF is not recommended to use because the member's cell position is not equally spread. Some cell position has a lot of registered members and the rest of them will remain empty. It makes the storage is not efficiently used.

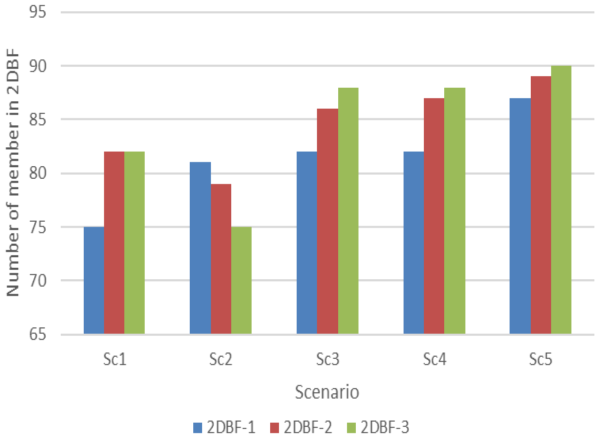

Figure. 6 The number of registered members in a different version of 2DBF

The 2DBF-2 and 2DBF-3 (except Sc2) focuses on the expanding dimension rather than its Bloom filter vector or its third dimension. They use a different prime number combination for each scenario. As can be seen in Figure. 6, the number of registered members in the 2DBF-2 and 2DBF-3 are higher than 2DBF-1 except for Sc2. It may be caused by the third dimension or the size of Bloom filter vector is not a prime number. The value of the dimension in the 2DBF is depended on modulo operation. If it is not prime numbers, then the result may not be evenly distributed. In addition, the higher the third dimension, the greater number of empty cells is, even though it is still lower than 2DBF-1.

Based on the latest experiment, there are some recommendations in determining the initialization parameter in the 2DBF:

1. avoid using the same value and make sure to use the prime number for every dimension;
2. focus on extending the first and second dimension;
3. select the appropriate scenario which is suitable for resource and environment.

The Sc5 of 2DBF-3 is the best scenario for the base station to recognize 90 members of sensor nodes. It is because this scenario has the lowest FPR and recognized all the members uniquely. This condition

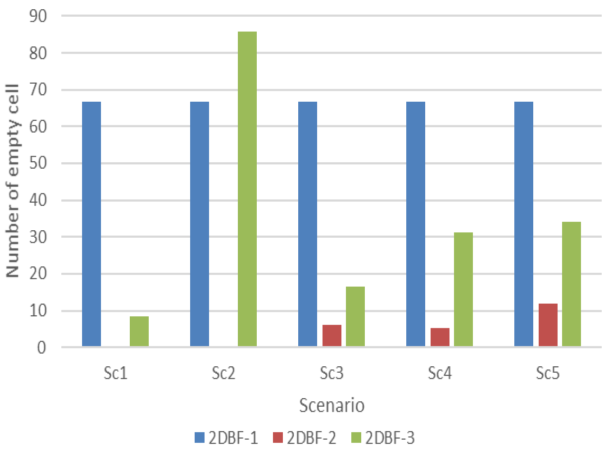

Figure. 7 The number of empty cells in the different version of 2DBF

is followed by high storage requirements and the number of empty cells. However, it does not become a problem because the Base station has more resources than sensor nodes.

The Sc1 of 2DBF-1 is the best scenario for a sensor node to recognize the other member. It is because this scenario needs the lowest storage requirement and the number of empty cells. It means this scenario uses the storage efficiently. It is suitable for the characteristic of sensor node which has a limitation in the storage and computation.
The initialization parameter recommendations are a general suggestion that can be implemented in the other study cases and the other number of members.

7. Conclusion

A membership scheme for a massive number of sensor nodes is needed in industrial automation. They need a fast mechanism to recognize a valid member so the cost of production can be reduced. In this study, a scalable 2DBF is recommended for massive scale WSN implementation. The experimental result shows that the computation complexity of 2DBF for insertion, query, and deletion is only O(1). However, the 2DBF's false positive probability decreased is only 33.06% among scenarios. Therefore, the proposed method has the lowest complexity although the false probability rate is higher than TinySet and ABF. This condition supports the 2DBF as the fastest procedure for a membership scheme. The storage usage of 2DBF is not only more efficient than the counting Bloom filter in average 131 bits difference but also approaches ABF in about 10 bits difference. Moreover, the analysis of the best initialization value has been conducted. This analysis result shows several recommendation rules for selecting the appropriate initialization parameter in a scalable 2DBF. These are general suggestions that can be used in other 2DBF case studies. Furthermore, several recommended scenarios can be analyzed for future works. There will be another optimization mechanism to measure the best scenario for a specific environment and to distribute members in a balanced cell position. It aims to improve 2DBF performance and increase efficient storage usage.

Conflicts of Interest

The authors declare no conflict of interest.

Author Contributions

Conceptualization, Farah and Cahya; methodology, Farah; software, Cahya; validation, Farah and Cahya; formal analysis, Farah; investigation, Farah; resources, Cahya; data curation, Cahya; writing—original draft preparation, Farah; writing—review and editing, Farah and Dedy; writing—draft finalization, Dedy; visualization, Farah; supervision, Farah; project administration, Farah; funding acquisition, Farah.

Acknowledgments

This work was supported by the Telkom University and Computing Laboratory under Intelligence System (IS) expertise group, School of Computing, Telkom University.

References

[1] C. Jehan and D. Shalini Punithavathani, "Cluster based trustable target detection and tracking scheme for wireless sensor networks", *International Journal of Intelligent Engineering and Systems*, Vol. 9, No. 4, pp. 205–214, 2016, doi: 10.22266/ijies2016.1231.22.

[2] D. R. Wijaya and F. Afianti, "Stability Assessment of Feature Selection Algorithms on Homogeneous Datasets: A Study for Sensor Array Optimization Problem", *IEEE Access*, Vol. 8, pp. 33944–33953, 2020, doi: 10.1109/ACCESS.2020.2974982.

[3] V. Talasila, C. Prasad, G. T. S. Reddy, and A. Aparna, "Analysis and prediction of crop production in Andhra region using deep convolutional regression network", *Int. J. Intell. Eng. Syst.*, Vol. 13, No. 5, pp. 1–9, 2020, doi: 10.22266/ijies2020.1031.01.

[4] M. Raza, N. Aslam, H. Le-Minh, S. Hussain, Y. Cao, and N. M. Khan, "A Critical Analysis of Research Potential, Challenges, and Future Directives in Industrial Wireless Sensor Networks", *IEEE Commun. Surv. Tutorials*, Vol.

20, No. 1, pp. 39–95, 2018, doi: 10.1109/COMST.2017.2759725.

[5] H. Xu, W. Yu, D. Griffith, and N. Golmie, "A Survey on Industrial Internet of Things: A Cyber-Physical Systems Perspective", *IEEE Access*, Vol. 6, pp. 78238–78259, 2018, doi: 10.1109/ACCESS.2018.2884906.

[6] S. Geravand and M. Ahmadi, "Bloom filter applications in network security: A state-of-the-art survey", *Comput. Networks*, Vol. 57, No. 18, pp. 4047–4064, 2013, doi: 10.1016/j.comnet.2013.09.003.

[7] K. Lin, H. Chen, T. Dai, D. Liu, L. Liu, and L. Shi, "Segmented Bloom Filter Based Missing Tag Detection for Large-Scale RFID Systems with Unknown Tags", *IEEE Access*, Vol. 6, pp. 54435–54446, 2018, doi: 10.1109/ACCESS.2018.2872543.

[8] X. Xie, X. Liu, H. Qi, and K. Li, "Fast Identification of Multi-Tagged Objects for Large-Scale RFID Systems", *IEEE Wirel. Commun. Lett.*, Vol. 8, No. 4, pp. 992–995, 2019, doi: 10.1109/LWC.2019.2903407.

[9] R. Patgiri, S. Nayak, and S. K. Borgohain, "PassDB : A password database with strict privacy protocol using 3D Bloom filter", *Inf. Sci. (Ny).*, Vol. 539, pp. 157–176, 2020, doi: 10.1016/j.ins.2020.05.135.

[10] F. Afianti, Wirawan, and T. Suryani, "Lightweight and DoS Resistant Multiuser Authentication in Wireless Sensor Networks for Smart Grid Environments", *IEEE Access*, Vol. 7, No. 1, pp. 67107–67122, 2019, doi: 10.1109/ACCESS.2019.2918199.

[11] S. Chen, M. Chen, and Q. Xiao, *Traffic Measurement for Big Network Data*. Springer International Publishing, 2017.

[12] A. Singh, S. Garg, K. Kaur, S. Batra, N. Kumar, and K. K. R. Choo, "Fuzzy-Folded Bloom Filter-as-a-Service for Big Data Storage in the Cloud", *IEEE Trans. Ind. Informatics*, Vol. 15, No. 4, pp. 2338–2348, 2019, doi: 10.1109/TII.2018.2850053.

[13] G. Einziger, R. Friedman, and B. Manes, "TinyLFU : A Highly Efficient Cache Admission Policy", *ACM Trans. Storage*, Vol. 13, No. 4, pp. 1–31, 2017.

[14] B. H. Bloom, "Space/ Time Tradeoffs in Hash Coding with Allowable Errors", *Commun. ACM*, Vol. 13, No. 7, pp. 422–426, 1970.

[15] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area Web cache sharing protocol", *IEEE/ACM Trans. Netw.*, Vol. 8, No. 3, pp. 281–293, 2000, doi: 10.1109/90.851975.

[16] G. Einziger and R. Friedman, "TinySet - An access efficient self adjusting bloom filter construction", *IEEE/ACM Trans. Netw.*, Vol. 25, No. 4, pp. 2295–2307, 2017, doi: 10.1109/TNET.2017.2685530.

[17] R. Ben Basat, "Poster Abstract : A Sliding Counting Bloom Filter", pp. 1012–1013, 2017.

[18] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom", In: *Proc. of the 10th ACM International on Conf. on emerging Networking Experiments and Technologies*, pp. 75–88, 2014, doi: 10.1145/2674005.2674994.

[19] S. Shomaji, F. Ganji, D. Woodard, and D. Forte, "Hierarchical Bloom Filter Framework for Security, Space-efficiency, and Rapid Query Handling in Biometric Systems", In: *Proc. of 2019 IEEE 10th International Conf. on Biometrics Theory, Applications and Systems (BTAS)*, No. October, pp. 1–8, 2019, doi: 10.1109/BTAS46853.2019.9185977.

[20] A. Singh, S. Garg, S. Batra, N. Kumar, and J. J. P. C. Rodrigues, "Bloom filter based optimization scheme for massive data handling in IoT environment", *Futur. Gener. Comput. Syst.*, Vol. 82, pp. 440–449, 2018, doi: 10.1016/j.future.2017.12.016.

[21] R. Patgiri, S. Nayak, and S. K. Borgohain, "rDBF : A r-Dimensional Bloom Filter for massive scale membership query", *J. Netw. Comput. Appl.*, Vol. 136, No. March, pp. 100–113, 2019, doi: 10.1016/j.jnca.2019.03.004.

[22] D. Blouin and S. Gordon, "A proxy-based Passive Duplicate Address Detection protocol for IPv6 wireless sensor networks", In: *Proc. of - 5th IEEE International Conf. on Control System, Computing and Engineering, ICCSCE 2015*, No. November, pp. 248–253, 2016, doi: 10.1109/ICCSCE.2015.7482192.

[23] D. Johnson, A. Menezes, and S. Vanstone, "The Elliptic Curve Digital Signature Algorithm (ECDSA)", *Int. J. Inf. Secur.*, Vol. 1, No. 1, pp. 36–63, 2001, doi: 10.1007/s102070100002.

[24] Y. Liu, J. Li, and M. Guizani, "PKC based broadcast authentication using signature amortization for WSNs", *IEEE Trans. Wirel. Commun.*, Vol. 11, No. 6, pp. 2106–2115, 2012, doi: 10.1109/TWC.2012.032812.110433.

[25] D. Eastlake and P. Jones, "RFC 3174-US Secure Hash Algorithm 1 (SHA1)(2001)", 2001.

[26] K. Ren, S. Yu, W. Lou, and Y. Zhang, "Multi-user broadcast authentication in wireless sensor networks", *IEEE Trans. Veh. Technol.*, Vol. 58, No. 8, pp. 4554–4564, 2009, doi: 10.1109/TVT.2009.2019663.

[27] F. Grandi, "On the analysis of Bloom filters", *Inf. Process. Lett.*, Vol. 129, No. September, pp. 35–39, 2018, doi: 10.1016/j.ipl.2017.09.004.