Node Wake-Up via OVSF-Coded Bloom Filters in Wireless Sensor Networks

Mirco Schönfeld and Martin Werner

Ludwig-Maximilians-University Munich, Germany,
mirco.schoenfeld@ifi.lmu.de,martin.werner@ifi.lmu.de

Abstract. Interest dissemination in constrained environments such as wireless sensor networks utilizes Bloom filters commonly. A Bloom filter is a probabilistic data structure of fixed length, which can be used to encode the set of sensor nodes to be awake. In this way an application can disseminate interest in specific sensor nodes by broadcasting the Bloom filter throughout the complete wireless sensor network. The probabilistic nature of a Bloom filter induces false positives, that is some sensor nodes will be awake without the application having interest in their sensor values. As the interest is often depending on location such as in adaptive sampling applications, we present a novel method to encode both interest and possible location of information into one probabilistic data structure simultaneously. While our algorithm is able to encode any kind of tree-structured information into a fixed length bit array we exemplify its use through a wireless sensor network. In comparison to traditional Bloom encoding techniques we are able to reduce the overall number of false positives and furthermore reduce the average distance of false positives from the next true positive of the same interest. In our example this helps to reduce the overall energy consumption of the sensor network by only requesting sensor nodes that are likely to store the requested information.

Key words: Bloom Filter, Location-Awareness, Sensor Networks

1 Introduction

In wireless sensor network (WSN), interest dissemination techniques play a central role in energy optimization. The sensor data of individual sensors of a wireless sensor network is often of varying importance for a given application. While in general all sensor information is needed, it is often not needed at any time and any place. Furthermore, techniques such as adaptive sampling give hints on areas, where a sensor network should acquire more data than in other areas. A common assumption about wireless sensor nodes is that they have only limited energy. The fact that a specific sensor information is of different importance for an application can then be used to reduce energy consuption and enhance network lifetime. The most simple way of reducing energy is to define timeslots and an algorithm, which defines whether a specific node should be awake in a given timeframe or whether it can power off. In simple sensing tasks, this form of energy reduction can be done locally by for example powering off the communication unit until a relevant change in a sensor variable is detected. However, the network can not adapt to new sensing tasks in this timeframe. More complex systems define a coordination timeframe in which all sensor nodes wait for new instructions for new sensing tasks. Inbetween two such coordination rounds it is then possible for many nodes to sleep and conserve energy.

But during a coordination round, how should the sensor network application communicate its current and future sensing interest? In this area, many approaches have been defined. A very convenient and often applied methodology is to employ a Bloom filter for interest dissemination. A Bloom filter is a probabilistic data structure, which is able to encode a set into a fixed length bit array. The central property of Bloom filters is that they only produce false positives. Hence, it is possible to encode a set using less bits than a traditional encoding without errors would need such that the encoded relation decodes to a superset. The price of this encoding is the fact, that the Bloom filter does not allow for removing elements from a filter.

With this paper, we propose a probabilistic data structure similar to a Bloom filter. However, our data structure incorporates a tree structure, which can be used for location-based applications. In short, our filter allows for inserting elements at specific locations, where a location is a positition in a binary tree and testing, whether a specific element has been added to a specific location before. This all can be encoded into a fixed length bit array which can be distributed in the coordination round of a wireless sensor network to disseminate the current location-dependent sensing interest throughout the network.

The central result of this paper is a coding structure, which allows for generalization and has local false positives. Generalization means, that adding an element at a specific place also adds this element at all parent nodes. In spatial indexing trees this results in adding the element to a specific place and all larger places containing this specific place. Furthermore, our technique generates local false positives in the following sense: False positives due to a wrong place (e.g., the element has been added, but in another place) have higher probability of being near to the right place, where the element has been added, as compared to a traditional Bloom encoding.

The rest of the paper is structured as follows: In Section 2, we briefly explain the classical Bloom filter and give some selected properties. Furthermore, we explain, how Orthogonal Variable Spreading Factor (OVSF) codes are constructed and which properties they have. In Section 3, we combine traditional Bloom filters with OVSF-codes to generate a location-aware Bloom filter. In Section 4, we discuss the properties of the newly proposed probabilistic set encoding. The following Section 5 shows applicability for interest dissemination and energy consumption in a wireless sensor network. Section 6 concludes this paper.

2 Hierarchical Bloom Overview

In traditional Bloom filter techniques a predefined number of hash functions is used to perform membership operations. When encoding both interest and location of information into one Bloom filter this is a strong limitation.

This section covers a detailed description of our approach which is able to circumvent this limitation by utilizing special characteristics of OVSF codes. To make this paper self-contained and to illustrate our solution we give a short introduction to Bloom filters before continuing with a description of our algorithm.

2.1 Hashing in Traditional Bloom Filters

A Bloom filter is a space-efficient data structure that represents a set of arbitrary data in order to support efficient execution of membership queries. Although Bloom filters allow false positives they have become very popular in database and networking applications. That is, because they allow for vast space savings and are very easy to use.

What Burton Bloom described with [2] in 1970 has not been changed since. He denoted a Bloom filter as a bit array of a fixed length m with all bits set to 0 initially. This array describes a set $S = \{x_1, x_2, ..., x_n\}$ of data containing n elements with $m << n$. Inserting an element x into S is followed by applying k independent hash functions $h_1, ..., h_k$ to that element and thereby calculating k different hash values $h_i(x)$ for $1 \leq i \leq k$. These hash values cause the corresponding bits in the filter to be set to 1. Therefore it is necessary that all relevant hash functions map all elements of the universe to a random number in the interval $\{1, ..., m\}$ and produce a uniform distribution over that interval.

Checking if an element y is in the set means calculating all hash values $h_i(y)$ for $1 \leq i \leq k$ and checking the corresponding k bits in the Bloom filter. If all bits are set to 1 the element is considered being in the set. Elsewise, if any bit is 0 the element is clearly not part of the set. While thereby excluding false negative responses Bloom filters are prone to yield *false positives*. That is, the filter states y being in the set even though it is not.

While one hash function h_i may map elements of the universe to a uniform distribution over the interval $\{1, .., m\}$ of bit adresses another hash function h_j may produce overlapping hash values as Figure 1 illustrates. So, one bit in the Bloom filter can not be related to one element of the set distinctively. Hence, Bloom filters yield false positives. This is the reason for the lack of supporting deletions, too. Assuming that removing an element from the Bloom filter means setting the corresponding bit adresses back to 0 it is unclear if the membership information of another element may be altered unwillingly.

The probability p of a Bloom filter producing a false positive response is calculated as follows. First, the probability p' for one bit still being 0 after n elements have been encoded into the filter is given. It is denoted as

$$p' = \left(1 - \frac{k}{m}\right)^n$$

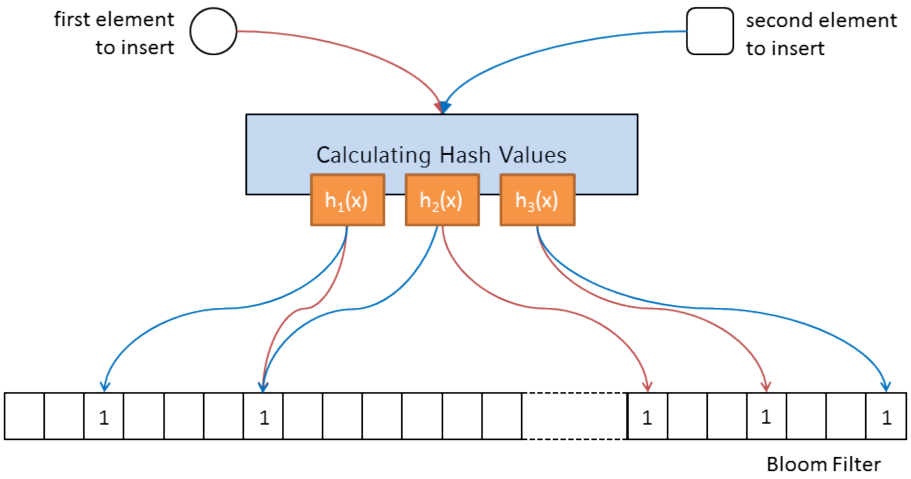


Fig. 1. Inserting different elements into a Bloom filter may result in overlapping bits to be set.

where k is the number of hash functions resulting in distinct bits set to 1 for each message in the given set. A false positive response is produced only if all k bits tested are 1. The probability p for this event is given by

$$p = (1 - p')^k. \tag{1}$$

This formula shows clearly how the probability of false positives is related to the filter size m, the size of the universe n and the number of hash functions k. A lot of research has been done focussing on choosing these parameters wisely as different applications rely on different false positive probabilities. Jardak et al. have given an analytical approach for wireless sensor networks (WSN) in [5] for example. Other interesting work concerned with optimal choice of parameters can be found in [3], [6], in [1] or even in [4].

In summary, the lower the false positive rate the better. But, the bigger the universe the higher the false positive probability for simliar filter sizes. Increasing filter size is one way to countersteer increasing false positive probability resulting in more space being needed. Chosing a well adapted (or even perfect) family of hash functions is another way which often results in adapting the hash functions for each element being added to the universe. Especially in WSNs, both countermeasures for increasing false positive probability are suboptimal where on one hand space is limited by small bandwith and on the other hand computing power is limited due to energy-efficient but low-scale hardware being used.

As stated earlier, we assume a tree-structured architecture of a WSN in which nodes have to be activated to measure data and where Bloom filters are used to chose the relevant nodes. Hence, a high false positive probability would result in waking up wrong sensor nodes and therefore in energy being wasted. Meanwhile, the architecture should provide an interface to wake up specific sensors at a specific location. With a traditional approach the universe for a Bloom filter would be described as a Cartesian product of the set of sensors and the set of locations. For a growing number of sensors and/or locations this would dramatically increase the size of the universe and thus calling for bigger Bloom filters. With our approach of hierarchical hashing this can be avoided. The next section will describe how OVSF codes can help.

2.2 Incorporating OVSF Codes

OVSF codes are used as a Code Division Multiple Access (CDMA) implementation for transmitting data via radio signals. OVSF codes build a complete binary tree as shown in Figure 2. This tree provides a hierarchical structure we use for partitioning of a universe. Besides, OVSF codes have two important characteristics our approach utilizes. Namely

1. **orthogonality** and
2. support of **generalization**.

Thereby, OVSF codes enable our approach to encode any kind of information into a fixed-length bit array that can be structured by a binary hierarchy. That is, OVSF code trees may represent a semantic ontology as well as a routing graph or a companies' organisation chart. In this paper and for the sake of Wireless Sensor Networks we use an OVSF code tree to segment a geographic area. But, whatever structure is represented by an OVSF code tree the smallest unit of this segmentation is a leaf node of the tree while each inner node covers multiple of those segments. Alltogether, the nodes of the tree provide *locations* into which elements of the universe can be inserted.

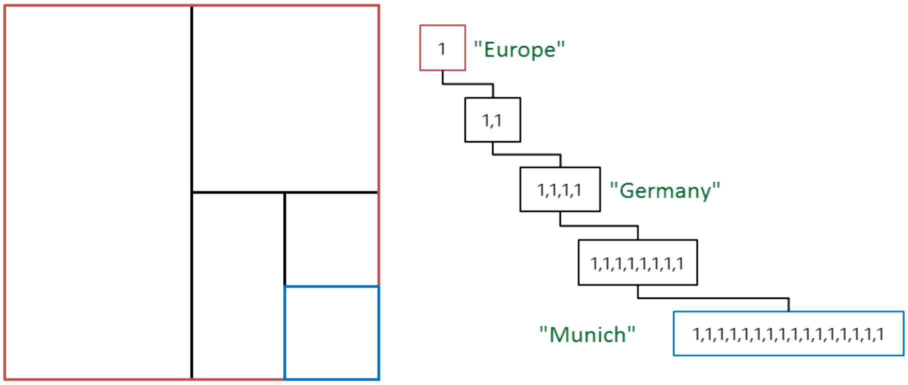


Fig. 2. An OVSF code tree and a corresponding segmentation of a geographic area.

In CDMA, codes of one level are assigned uniquely to the senders of a network. Meanwhile, all sibling OVSF codes are mutually orthogonal to each other meaning their scalar products equal 0. In signal processing applications, their orthogonality is used as two signals multiplied with two such sibling-codes are spread over a wide frequency spectrum and hence do not interfere anymore. In our approach, we utilize the orthogonality of OVSF codes to produce a unique combination of hash functions for each location an element of a universe could be inserted at.

Another advantage of OVSF codes is their immanent support of generalization. That is, each node in the tree contains the complete code of all of its parent nodes because each subsequent level can be calculated from an active level in a recursive manner. The algorithm to calculate both codes $c_{1,l+1}$ and $c_{2,l+1}$ of level $l+1$ is also denoted as

$$\begin{aligned} c_{1,l+1} &= [c_l, c_l]\,, \\ c_{2,l+1} &= [c_l, !c_l]\,. \end{aligned} \tag{2}$$

How this is used to add only distinct functions to the set of hash functions when descending the tree towards the leaf nodes is part of the following section. However, inserting an element at a location of a leaf node means it is automatically inserted at all parent locations by applying their hash functions, too.

3 Hierarchical Hashing

In this Section we present a novel approach to encode both interest and possible location of information into one probabilistic data structure simultaneously. For that purpose, we let location information be represented by some binary spatial indexing tree and generate a binary OVSF-tree of equal size. Of course, any other binary tree structure of *location* can be used.

The main idea is to let the OVSF code control the set of hash functions to be activated during insertion of an element at a given location. Therefore, inserting an element into hierarchical Bloom filter now requires the element itself together with the OVSF code of the location where the element is to be inserted at.

For the following, let us assume some finite OVSF-tree. An *OVSF control code* is then the OVSF-code from the tree padded from the left with zeros, such that the OVSF control code is an array c_l of some fixed length k and each digits value is one of $\{0, 1, -1\}$ Let us now chose $2k$ independet hash functions $H = \{h_1^+, ..., h_k^+\} \cup \{h_1^-, ..., h_k^-\}$ mapping the universe to the integer interval$\{1, ..., m\}$ following a uniform distribution. Inserting an element e at a location given by an *OVSF control code* is then done by traversing the control code. For a 0 in position i of the control code, nothing is done. For a 1 in position i of the control code, we activate the hash function h_i^+ and for a -1 in position i of the control code, we activate the hash function h_i^-. Activation of a hash function means hashing the element e to be inserted with the hash function and using the result as an index into a fixed length filter array and set the corresponing entry to 1.

Using this algorithm, a shorter OVSF-code activates fewer hash functions and codes, which are children in the OVSF code tree activate the same hash functions and some additional ones. As a result, this algorithm allows for generalization. If an element has been added to some place corresponing to some OVSF code, then all hash functions used by parent OVSF control codes have also been activated, such that an element test after insertion returns true for all parent locations. In other words, adding an element to a low level of an OVSF tree means adding it to all parent levels first by applying all parent level's hash functions and then calculating additional hash values to distinguish the resulting filter pattern from a pattern produced by a sibling's location code.

Furthermore, we expect this algorithm to generate local false positives due to the following fact: The nearer two codes are in the OVSF tree structure the longer is the common prefix. That means, that two elements, which are near to each other inside the tree, activate the same hash functions and some additional ones. Hence, the false positive probability is higher as compared to distant nodes, which activate many different hash functions.

Another property of OVSF-codes plays off here: While two sibling nodes of the same length have a long common prefix, the construction of the OVSF-code makes the rest of the code completely different. Hence for two codes, which share a long prefix, the OVSF-construction leads to activation of a disjoint set of independent hash functions. This puts some limit on the false positive rate with respect to siblings.

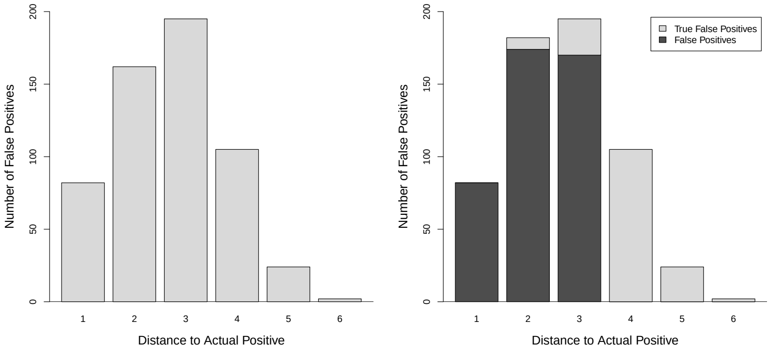
As a summary, let us give some results about the algorithm, which are clear from the construction.

1. An element e inserted at a location code c is also inserted to the locations associated with all parent codes.
2. The distinguishability of two sibling nodes is maximal in the following sense: As parents shall return positives, all parent code hash functions have to be activated. But for two siblings, these are all common hash functions. The rest of the hash functions is a disjoint set of independent hash function leading to minimal false positives between siblings.
3. The number of hash functions to be used is increasing exponential with the OVSF tree depth. This is the main drawback of the construction.

In the following section, we will evaluate the complete approach giving number to the locality of false positives and also compare this approach to a straightforward application of traiditional Bloom filtering. For this comparison, we insert the concatenation of the location code and the element into a set of hash functions and activate them. To allow for comparison of results, we choose the Bloom filter parameters in a way, such that the same number of hash operations is used.

4 Data Structure Evaluation

As a proof of concept we inserted an element $u \in U$ at a OVSF-location $c_l \in C$ into a Bloom filter and then checked the filters' response for every element $u' \in U$ being at every possible location $c'_l \in C$. For each false positive filter response we identified the shortest path through the OVSF code tree between location c'_l the



(a) False positive distances d_{fp} (b) True false positive distances d_{fp_t}

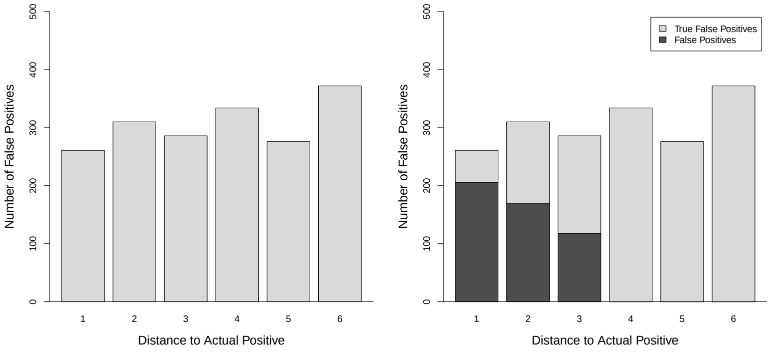
Fig. 3. Histograms of false positive and true false positive distances with Bloom size of $m = 16$, $|U| = 7$ and $|C| = 15$

element was reported to be at and location c_l the element was actually inserted at. We denote this shortest path as the *false positive distance* d_{fp}.
Each false positive response was further analyzed by checking if location c'_l was a parent location of c_l. If so, the false positive was expected and desired since that response was considered being the effect of generalization of OVSF codes. If c_l was not any of c'_l direct or indirect child nodes the distance between locations c'_l and c_l was measured as d_{fp_t} a *true false positive distance.*
We subsequently repeated this for inserting every $u \in U$ at every location $c_l \in C$ while checking all combinations of elements and locations again.

Preparations of our experiments included generating an OVSF tree with the algorithm given in equation 2, first. Also, insert operations were implemented as proposed in Section 3. All experiments were conducted in GNU Octave.

As a start we chose a universe U to contain 7 different elements and generated an OVSF tree C with 4 levels and thereby 15 possible insert-locations c_l. Since an OVSF code c_l on level $l = 4$ has 8 digits our set of hash functions consists of 8 independent functions h_i with $i = 1..8$. Every h_i uses the well-known md5 hashing algorithm each with different seeds. When an element is inserted to a leaf location all 8 hash functions are applied to the element resulting in a bit pattern with 8 bit set. Therefore, we chose a Bloom filter with $m = 16$ bits to be suitable.

Figure 3(a) shows a resulting histogram of false positive distances for the mentioned experimental setup. Its pyramid-like shape is a promising result since the hierarchical structure of the OVSF code tree has been successfully mapped onto the one-dimensional Bloom filter, apparently.
The histogram clearly has a peak at distance $d = 3$ which equals the length of the shortest path from a leaf node to the root node in an OVSF code tree of height 4. Every distance $d > 3$ means the path between the actual and reported locations definitely crossed the root node while this is not always the case for $d = 3$ or $d = 2$.



(a) False positive distances d_{fp} (b) True false positive distances d_{fp_t}

Fig. 4. Histograms of false positive and true false positive distances with traditional inserting strategies for a Bloom size of $m = 16$, $|U| = 7$ and $|C| = 15$

Therefore, Figure 3(b) depicts the fraction of true false positives that where measured when the filter reported a positive result even though the element was inserted into a sibling branch. While the effect on all bins for $d > 3$ was expected, there were a few true false positives for $d = 2$ and $d = 3$. For $d = 2$ the very low number of true false positives may be explained with the orthogonality of OVSF codes of the same level since $d = 2$ steps always lead to the direct sibling.

Additional experiments have been conducted to compare our approach to traditional inserting strategies. There are a few obstacles, though. Traditional techniques do not provide any way to link an interest in specific information together with a possible location of this desired information. Also, they always require a fixed number k of hash functions being applied.

To simulate a link of interest and location we concatenate the location description and the element being inserted. To approximate the number k of required hash functions, we first calculate the average number of hashing operations needed for structured insertions and let this serve as a k for our implementation of a conventional technique.

For the proposed setup with an OVSF tree of height 4 a code corresponding to a leaf node has 8 digits while the code representing the root node has only 1. On average, our approach uses 5.66667 hash operations for each insertion which is why we set $k = 5$.

Figure 4(a) shows the resulting distance histogram for false positive distances d_{fp} of a conventional insertion strategy. Compared to Figure 3(a) the high number of false positives stands out. Also, the histogram is not indicating any distinct tendency while in fact showing high random-like fluctuation between bins. This impression changes a little when considering Figure 4(b) where the distances d_{fp_t} of true false positives have been integrated into the histogram. Again, all bins for $d > 3$ show only true false positives which was expected. What is remarkably though is the remaining false positives for $d \leq 3$ having the tendency to decrease towards $d = 3$. This is actually a desired quality since false positives should accumulate near to the actual positive. Especially, when compared to Figure 3(b) where the false positives increase towards the root node this could be an advantage over our approach. However, this quality vanishes among the high overall false positive rate.

Table 1 holds a summary of the proposed results. The element count represents the overall number of membership tests of elements in our experiment. Since each of the $|U| = 7$ elements being inserted at $|C| = 15$ different locations produced one distinct Bloom filter that was queried with each combination of elements and locations one experiment contained $(7*15)*(7*15) = 11025$ membership tests. Our approach of hierarchical hashing produced 576 false positives of which 170 were true false positives where the element was inserted nowhere along the tested branch. That is a false positive rate of 5.2245 compared to a false positive rate of 17.7687 of traditional hashing. Only considering the true false positives our approach has a rate of 1.542 compared to 13.288. In our approach 29.51% of false positives were true false positives while in traditional hashing true false positives make up a fraction of 74.78% of all false positives. Furthermore, we were able to reduce the average distance of a true false positive to 3.78 steps through the OVSF code tree.

Table 1. Summary of Distance Histogram Results for Bloom size of $m = 16$, $|U| = 7$ and $|C| = 15$

	Hierarchical Bloom	Traditional Bloom
Element count	11025	11025
False Positives	576	1959
False Positive Rate	5.224490%	17.768707%
True False Positives	170	1465
True False Positive Rate	1.541950%	13.287982%
True False Positive Distance	3.782353	3.9501
Number of Calculated Hashes	595	540
Average Number of Calculated Hashes	5.666667	5.142857143

One explanation of the high false positive rate of traditional hashing may be found in using the concatenation of the element and the location as a parameter of hash functions. This concatenation equals a Cartesian product of the set of elements and the set of locations and thereby dramatically increasing the size of the universe. Adapting false positive rates of hierarchical hashing for traditional techniques requires different filter parameters. That is, a bigger filter size for example.

This should emphasize how our approach of hierarchical hashing helps to reduce the required filter size while improving the false positive rate of the filters' parameter settings. Furthermore, we are able to concentrate false positive responses closer to the actual positive while also reducing the probability of descending a wrong branch of the tree.

The following section describes a prototypical implementation of a wireless sensor network (WSN) that incorporates our technique of hierarchical hashing.

5 Application to Sensor Network Management

We evaluate the hierarchical hashing approach for Bloom filters using a sample Wireless Sensor Network (WSN) architecture. This architecture is characterized by several sensors being available at the same location and being seperated from each other. Thus, each sensor can be activated seperately which can be useful in several scenarios. In a crowd-sourced mobile phone sensing approach for example nodes can request certain sensor information from each other such as one node needs information about loudness from a certain participator while another node urges nitrogen concentration from that very participator. Both requests can be encoded into one single Bloom filter which is forwarded to a cluster head of the relevant location. The cluster head then extracts information which sensor of which of its succeeding nodes are to be activated.

Another scenario is imaginable where a traffic monitoring system is deployed to roadside units (RSUs) such as traffic lights. These RSUs can request certain information from passing vehicles such as vehicle's average speed or information about traffic density. Therefore, the RSU encodes the information request into a single Bloom filter which is sent to a vehicle nearby. The vehicle extracts details of the information request and responses with relevant data.

Regardless of specific details, both scenarios incorporate a hierarchical structure at some point. On one hand, the participatory sensing framework has its hierarchicy given by the network's topology combining nodes into clusters and designating cluster heads. The traffic monitoring system on the other hand, consideres sensors of passing vehicles being hierarchically structured. In that scenario, every vehicle has an internal monitoring system which receives a Bloom filter and then gathers relevant information from the vehicle's data collection instances.

In both scenarios every node except leaf nodes needs to know what information can be collected from which of its succeeding branches. A more detailed knowledge is not necessary since every succeeding node is responsible for proper forwarding of requests.

5.1 Description of Wireless Sensor Network Architecture

This evaluation uses a generic WSN in which nodes are geographically distributed over a certain area. The OVSF code tree partitions this area into several segments providing a hierarchical overlay structure for the area.

In our generic scenario requesting information from a specific source from a specific location means to insert the information source together with the OVSF code corresponding to the relevant location into a Bloom filter following the proposed algorithm. This Bloom filter is sent to the root node of the network.

On receiving a Bloom filter the root node checks which sensors the root itself should activate, first. Afterwards, it analyzes which sensors from which of its succeeding branches should be activated forwarding the Bloom filter only to relevant children. This sequence continues for each of the child nodes receiving the Bloom filter.
In concrete application frameworks this generic algorithm could be easily adopted to letting the inner nodes decide which of the children's sensors should be activated allowing deployment of cheap and highly energy efficient sensors on the lowest level of the hierarchy.

To evaluate the network's performance we measure energy consumption of each active sensor. In comparison to traditional inserting strategies we expect the overall energy consumption to be lower due to lower true false positive rate. Nevertheless, if a wrong sensor is being activated it is, in theory, closer to the requested sensor due to lower distance d_{fp_t} of true false positives.
For evaluation of energy consumption we implemented a framework that is capable of counting both the expected and the actual energy consumption of each sensor in the network. Therefore, our framework simulates a WSN for a predefined number of rounds. In every iteration a number of sensors is chosen to be activated. These sensors that are to be activated at a specific location are then inserted into a Bloom filter. While the maximum number of simultaneous activations is restricted the actual number of activations is chosen randomly between 0 and that threshold.
For every iteration t of the simulation two states are kept. Namely, both the expected state that contains the added energy consumption for each sensor up to that t and a second state that contains the actual energy consumption for each sensor that was activated from a Bloom filter. Thereby, the second state contains all sensors that were activated by mistake meaning after a false positive response of the Bloom filter.
This enables us to evaluate the performance of a Bloom filter in such an application by comparing the optimal and the actual energy consumption of the network. Also, we are able to compare the hierarchical insertion strategy and the traditional one. The following section gives results of our conducted experiment.

5.2 Energy Consumption Evaluation

As a proof of concept we simulated a small WSN environment with 15 different locations each providing a maximum of 4 different sensors. In total we randomly distributed a number of 32 sensors into this structure which corresponds to an average of 2.13 sensors per location. For simplification, every sensor consumed 1 unit of energy in every round it was active. We stated that 5 sensors could be activated simultaneously with one Bloom filter of 16 bits. Once activated, a sensor remained active for 3 rounds. The simulation time was 150 rounds during which 6750 membership queries were executed in total. Table 2 summarizes this setup again in detail.

Table 2. Summary of Experimental Setup

	Values
Filter size in bits	16
OVSF tree depth (and location count)	4 (15)
Number of sensors distributed over the net	32
Average number of sensors per location	2.13
Rounds	150
Maximum number of activations per Round	5
Number of rounds for sensors to remain active	3
Number of Bloom membership queries	6750

The simulation's results are combined in Table 3. Clearly, our hierarchical insertion strategy performs better than a comparable traditional approach. The false positive rate being much lower results in a considerably lower energy consumption. False positive rates were calculated with the total of 6750 Bloom membership queries. However, both approaches are suffering from a high false positive rate leading to a high degrading factor of energy consumption. The degrading factor describes the relation between optimal and actual energy consumption values. A factor > 1 means there was more energy used than expected. According to the results our hierarchical hashing approach consumes 7.5 times more than optimal while the traditional technique performed even worse with a factor of 13.77. However, in comparison to a traditional approach our hierarchical hashing algorithm reduced the amount of energy wasted through falsly activated sensors by 65%.

Table 3. Summary of Evaluation in WSN Example Application

	Hierarchical Bloom	Traditional Bloom
False Positives	1508	3626
False Positive Rate	22.34%	53.72%
Actual vs. Optimal Overall Energy Consumption	2903 / 321	4421 / 321
Degrading Factor of Energy Consumption	7.512428	13.7726
Average Number of Calculated Hashes Per Node	143.133	388

Figure 5 shows a histogram of average false positive distances for both insertion techniques. While the traditional procedure producing more false positives evidently both show a similar tendency of having a peak at distance $d = 3$. Especially for the traditional approach this seems to contradict results shown in Figure 4(b). But, it is to notice that in contrast to Figure 3(b) and 4(b) this histogram contains average distances instead of actual ones. That is, because the proposed framework activated several sensors at once with one Bloom filter. But several elements being inserted into one filter preclude an evaluation of distances between false positive filter responses and the actual elements and thereby eliminating distinguishability between false positives and true false positives.

It can be concluded from Figure 5 that the hierarchical approach outperforms a traditional one with regard to the amount of false positives. In addition, the hierarchical approach seems to lack the tendency of false positives being reported with a distance $d > 3$ meaning that false positives are more likely to occur within the actual branch towards the root node.

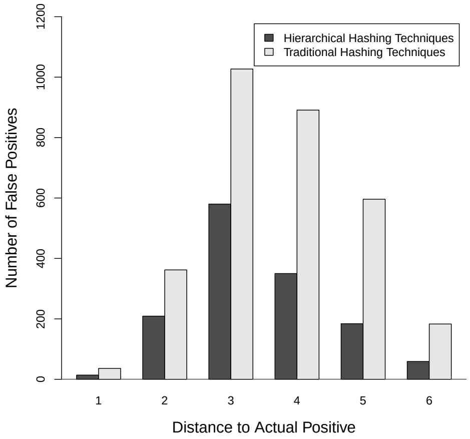


Fig. 5. Histogram of Average Distances of False Positives

Figure 6 depicts the number of requested and actually activated sensors for every iteration of the simulation. While both the traditional and our hierarchical approach always activate more sensors than necessary the hierarchical technique seems to better adapt to the actual requests. Predominantly, the development of requests can be roughly identified in the development of activations caused by hierarchical hashing. In contrast, the traditional insertion strategy causes every of the 32 sensors to be activated frequently which is why its graph seems to be cut off at that value.

Numerous unrequested activations are cause by a high false positive rate. The false positive rate has been further analyzed and is shown in Figure 7(a). The histogram depicts the false positive rate for each location. The root node of the OVSF tree is shown at $x = 1$ while its direct successors are inserted at $x = 2$ and $x = 9$. Children of $x = 2$ on the other hand are located at $x = 3$, $x = 6$ and children of $x = 9$ can be found at $x = 10$ and $x = 13$, respectively.

With this mapping of the tree structure in mind, it is observable for our hierarchical hashing how false positives diminish further down the tree. This behavior corresponds to evaluations of figures 3(a) and 3(b). Accordingly, there is no clear relation between false positives and their location in a OVSF code tree for a comparable traditional hashing procedure.

In turn, numerous unrequested activations result in a bad energy balance, understandably. The relative energy balance of each OVSF location was further analyzed and is shown in Figure 7(b), where the degrading factor for each lo-

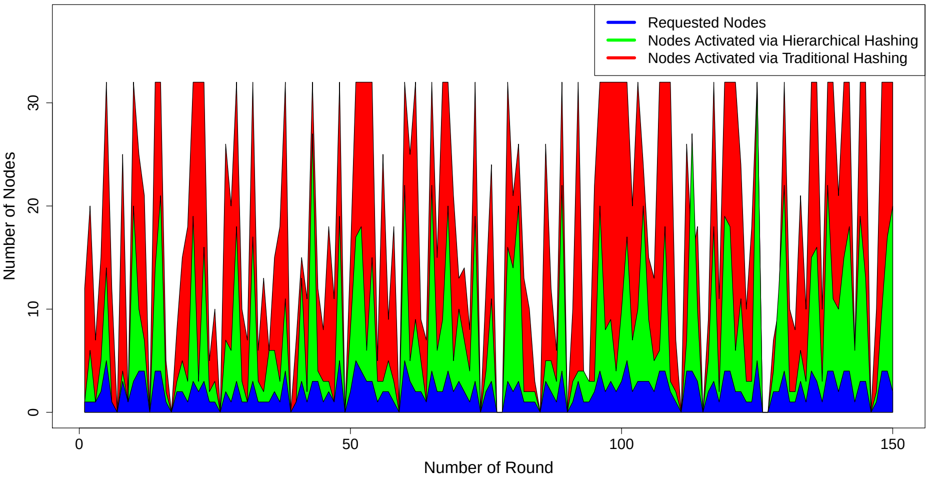
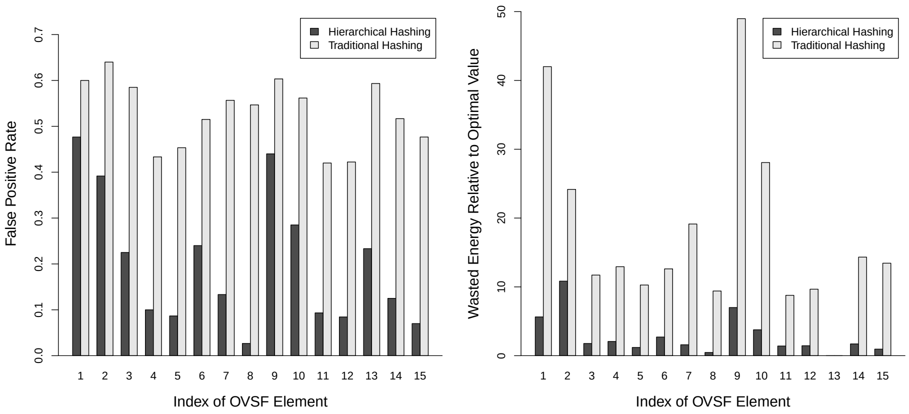


Fig. 6. Number of Requested Sensors Compared to the Number of Actually Activated Sensors



(a) False positive rate

(b) Energy consumption at each location relative to its optimal value

Fig. 7. False positive rate and energy consumption at each location

cation is given. A factor > 1 means that energy is being wasted. Both charts showing a gap at location $x = 13$ is correct as that location did not provide any sensor.

Our hierarchical approach wasting less energy compared to the traditional approach is unsurprising because both false positive rate and average degrading factor of the compared technique are noticeably higher. What is remarkably though is the degrading factor for the root node at location $x = 1$. It could have been expected from Figure 7(a) that the degrading factor is particularly high at this location as the Figure stated the highest false positive rate here.

Additionally, the root node processes every incoming Bloom filter and only one bit in the Bloom filter is necessary to activate sensors here.[1]
Contrary to this expectation, the degrading factors are considerably higher for the root's direct successors at $x = 2$ and $x = 9$. This leads to the conclusion that sensors provided at the second level of a OVSF code tree are prone to false activations while this sensitivity diminishes for lower levels.

6 Conclusion & Future Work

In this paper we proposed a novel hashing algorithm that enables us to successfully encode a hierarchically structured set of data into a one-dimensional probabilistic bit array. We have given a broad evaluation of our method in a generic setup where an OVSF code tree was utilized to structure a set of data and a Bloom filter served as the probabilistic bit array. In comparison to traditional insertion strategies a low false positive rate in relation to the tree structure emerged for our approach. Subsequently, we proposed an architecture of a WSN to evaluate our algorithm's performance in a real-world example application. Therefore, the WSN framework utilized our hierarchical hashing method to activate specific sensors at specific locations. We were able to reduce the amount of energy wasted through falsly activated sensors by 65%.
In future work we will give an analytical proof of our method. Also, further analysis of optimal Bloom filter parameters and utilization of different hierarchical structures will follow.

References

1. Paulo Srgio Almeida, Carlos Baquero, Nuno Preguia, and David Hutchison. Scalable bloom filters. *Information Processing Letters*, 101(6):255 – 261, 2007.
2. Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
3. Prosenjit Bose, Hua Guo, Evangelos Kranakis, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel Smid, and Yihui Tang. On the false-positive rate of bloom filters. *Information Processing Letters*, 108(4):210 – 213, 2008.
4. Sarang Dharmapurikar, Praveen Krishnamurthy, Todd S Sproull, and John W Lockwood. Deep packet inspection using parallel bloom filters. *Micro, IEEE*, 24(1):52–61, 2004.
5. C. Jardak, J. Riihijarvi, and P. Mahonen. Analyzing the optimal use of bloom filters in wireless sensor networks storing replicas. In *Wireless Communications and Networking Conference, 2009. WCNC 2009. IEEE*, pages 1–6. IEEE, 2009.
6. Michael Mitzenmacher. Compressed bloom filters. *IEEE/ACM Trans. Netw.*, 10(5):604–612, October 2002.

[1] That is, because the root node's OVSF code is precisely one bit long.