Detection and Correction the Decayed Bits of Scope Decay Bloom Filters In Wireless Sensor Networks (Wsns)

Shamim Ahmed*, Ishtiaque Mahmud, A.K.M. Nazmus Sakib***, Md. Habibullah Belali****, Sajeeb Saha*****,Quazi Emanual Alendey********

*(Completed B.Sc. in Computer Science and Engineering from Dhaka University of Engineering and Technology (DUET), Gazipur, Bangladesh.)

**(Completed M.Sc. and B.Sc. in Computer Science and Engineering from Jahangirnagar University (JU), Dhaka, Bangladesh.)

***(Completed B.Sc. major in Computer Science and Engineering from Chittagong University of Engineering & Technology (CUET), Chittagong, Bangladesh.)

****(Completed B.Sc. in Computer Science and Engineering from University of Dhaka (DU), Dhaka, Bangladesh,)

*****(Completed M.Sc. and B.Sc. in Computer Science and Engineering from University of Dhaka (DU), Dhaka, Bangladesh,)

******(Completed B.Sc. in Computer Science and Engineering from Jahangirnagar University (JU), Dhaka, Bangladesh.)

ABSTRACT

In WSNs the existing query flooding based and event flooding based routing protocol, Query flooding can find desired events quickly but is also costly because many query messages are generated and employs precise routing hints to route queries that can reduce query messages at the expense of heavy routing overhead (specifically, keeping precise routing hints for many events is expensive) respectively. Bloom filters have been used in database applications, web caching, and searching in peer-to-peer networks. In this paper, we propose a routing protocol in Wireless Sensor Networks (WSNs) by *Scope Decay Bloom Filter (SDBF)*, that detecting and correcting the decaying bits of SDBF using Hamming Code. In SDBF, each node maintains some probabilistic hints about events and utilizes these hints to route queries intelligently. SDBF greatly reduces the amortized network traffic without compromising the query success rate and achieves a higher energy efficiency.

Keywords - Bloom Filters, Error Correction, Error Detection, Hamming Code, Routing Protocol, Scope Decay Bloom Filters, Wireless Sensor Networks (WSNs).

I. INTRODUCTION

The Bloom filter a way of using hash transforms to determine set membership [1]. Bloom filters find application wherever fast set membership tests on large data sets are required. Such applications include spell checking, differential file updating, distributed network caches, and textual analysis. It is a probabilistic method with a set error rate. Errors can only occur on the side of inclusion | a true member will never be reported as not belonging to a set, but some non-members may be reported as members.

A wireless sensor network is a collection of nodes organized into a cooperative network [2]. Each node consists of processing capability (one or more microcontrollers, CPUs or DSP chips), may contain multiple types of memory (program, data and flash memories), have a RF transceiver (usually with a single omni-directional antenna), have a power source (e.g., batteries and solar cells), and accommodate various sensors and actuators. The nodes communicate wirelessly and often self-organize after being deployed in an ad hoc fashion. Systems of 1000s or even 10,000 nodes are anticipated. Such systems can revolutionize the way we live and work. Wireless sensor networks (WSNs) have been used in applications such as the health industry, military, warehouse, and home environment [3]. Sensors are typically low-cost, low power, and multi-functional. They communicate with each other through wireless media and form a wireless distributed network. In WSNs, routing is data-centric, i.e. finding data with specific attribute values [4]. In many WSNs applications, routing is query-based. A *sink* initiates a query for some desired data, which is forwarded towards the hosting sensors [5]. Sinks can be static or dynamic.

In this paper, we propose a routing protocol in WSNs *by Scope Decay Bloom Filter (SDBF)*, that detecting and correcting the decaying bits of SDBF using Hamming Code and utilizes probabilistic hints. Each sensor maintains probabilistic hints about events that may be found through its neighbors. Hints are encoded using the proposed variant of the bloom filter (BF) [6] [7], called *scope decay bloom filter (SDBF)*.BF consists of a bit string and a group of hash functions. To generate a BF for a set, each set element is mapped by each hash function to a bit

position in the bit string. All mapped bits are set. To determine the membership of an item, the item is hashed similarly. If any of the hashed bits is not set, then the item definitely does not belong to the set. If all bits are set, then the item is *possibly* in the set. If in fact the set does not contain the item, a false positive occurs. Nevertheless, the space savings usually offset this shortcoming when the false positive rate is significantly low. Bloom filters have been used in database applications [6], web caching [8], and searching in peer-to-peer networks [9] [10].
The SDBF protocol uses routing hint about an event. The advertisement is designed such that the hint does not decay within the *k*-hop neighbourhood of an event source but decays outside the *k*-hop neighbourhood as the distance from the boundary of the *k*-hop neighbourhood increases. By trading off precise routing hints for probabilistic ones, SDBF achieves a higher query success rate with the same or less amortized routing overhead.

1.1. Novel Uses

This section reviews some of the most interesting applications of Bloom filters. It is perhaps surprising that what is essentially a set-membership test is of use in so many important applications.

- **Rule-based systems:** Burton H. Bloom originally proposed filter hashing as part of a program to automatically hyphenate words. He wanted to separate words that could be hyphenated by the application of simple rules from the minority that required extensive analysis. He proposed using his filter method to separate the 10% of difficult words from the rest [11].
- **Spell Checkers:** Bloom filters have been successfully applied in spell checking programs such as cspell [12][13][14]. They are employed to determine if candidate words are members of the set of words in a dictionary. In the case of cspell, suggested corrections are generated by making all single substitutions in rejected words and then checking if the results are members of the set [12]. Bloom filters perform very well in such cases [12]. The filter size was chosen to be large enough to allow the inclusion of additional words added by the user.
- **Estimating Join Sizes:** Mullin [15][16] suggested using Bloom filters to estimate the size of joins in databases. This is of particular advantage in the case of distributed databases where communications costs are to be kept to a minimum. He presented a method by which filters that are too large to fit in memory can be used [16]. The method is essentially to use a representative sample of a filter for testing and ignore all hashes outside the range of the sample. Since hash transforms are pseudo-random, any significantly large portion of a filter can act as a sample.
- **Differential Files:** A major area of interest in the application of Bloom filters has been their use in differential file access [13][14]. A differential file is essentially a separate file which contains records that are modified in the main file [13]. Differential files are used as caches in large databases: when a change is made to a record in the main database the differential file is updated; when all the changes have been made to the database then the differential file is used to update the database. When the differential file is much smaller than the database, changes to it can be made without the overhead needed to search the main file. Of course, it would be best to keep the entire set of records in memory at once, but this is not feasible for large data sets and so the probabilistic approach offered by Bloom filters is used. Bloom filters in core memory are used to predict if a record will be found in the differential file.

1.2. Related Works

The simplest way to route queries is to flood queries from the sink over the entire WSN and set up the reverse paths for desired data to be sent back to the sink. Various query flooding schemes differ in the manner in which they set up and use reverse paths. Directed diffusion [17] tries to find an optimal path between the sink and the event sources by flooding an exploratory query that is initiated at a sink. Each node sets gradients between neighboring nodes, and reinforces the best route for real data while transferring the exploratory events on the reverse query path. The gradients are only used for sending the real data from the discovered event source to the sink that initiates the exploratory query. Gradient-based routing [18] is another scheme based on query flooding. It associates each node with a height, which is the minimum distance in terms of the number of hops from the sink. The scheme also assigns a gradient to the link between a node and its neighbor. A gradient is defined as the height difference between a node and its neighbor. A node always forwards desired data through the link with the highest gradient among all links to its neighbors. Energy aware routing [19] is also based on query flooding. This scheme tries to maintain multiple paths between a data source and the sink. Desired data is propagated through a route that is probabilistically selected. The probability of a route is set based on its energy consumption. To reduce the cost of query flooding, gossiping [20] can be used for query-based routing in WSNs. It is essentially a random walk where each node forwards a received query to a randomly chosen neighbor.

II. STANDARD BLOOM FILTERS

The Bloom filter is used to test whether an element is a member of a set and is a space-efficient probabilistic data structure, false negatives are not possible but false positives are possible. Elements can only be added to the set and cannot be removed. The probability of false positives increases when more elements are added to the set.

An empty Bloom filter is a bit array of *m* bits, all set to 0. There must also be *k* different hash functions defined, each of which maps or hashes some set element to one of the *m* array positions with a uniform random distribution. To add an element, feed it to each of the *k* hash functions to get *k* array positions. Set the bits at all these positions to 1. The requirement of designing *k* different independent hash functions can be prohibitive for large *k*. For a good hash function with a wide output, there should be little if any correlation between different bit-fields of such a hash, so this type of hash can be used to generate multiple "different" hash functions by slicing its output into multiple bit fields. Alternatively, one can pass *k* different initial values (such as 0,1, ,,, k-1) to a hash function that takes an initial value, or add these values to the key. For larger *m* and/or *k*, independence among the hash functions can be relaxed with negligible increase in false positive rate. Removing an element from this simple Bloom filter is impossible. The element maps to *k* bits, and although setting any one of these *k* bits to zero suffices to remove it, this has the side effect of removing any other elements that map on-to that bit, and we have no way of determining whether any such elements have been added [21]. Such removal would introduce a possibility for false negatives, which are not allowed.

An example of Bloom filter is shown in Fig 1. This is representing the set {e1,*e2*}. The coloured lines show the positions in the bit array that each set element is mapped to. For this figure *m*=16 and *k*=3. Removal of an element from a Bloom filter can be simulated by having a second Bloom filter that contains items that have been removed. However, false positives in the second filter become false negatives in the composite filter, which are not permitted.

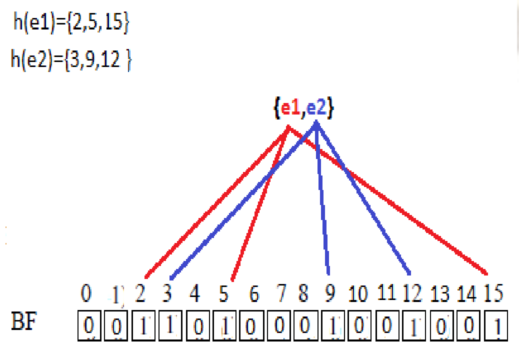

Fig.1: Standard Bloom Filter

This approach also limits the semantics of removal since re-adding a previously removed item is not possible. However it is often the case that all the keys are available but are expansive to enumerate. When the false positive rate gets too high, the filter can be regenerated. This should be a relatively rare event.

III. PROBABILITY OF FALSE POSITIVE

Assume that a hash function selects each array position with equal probability. If *m* is the number of bits in the array, the probability that a certain bit is not set to one by a certain hash function during the insertion of an element is then

$$1-\frac{1}{m}.$$

The probability that it is not set by any of the hash functions is

$$\left(1-\frac{1}{m}\right)^k.$$

If we have inserted *n* elements, the probability that a certain bit is still 0 is

$$\left(1-\frac{1}{m}\right)^{kn};$$

the probability that it is 1 is therefore

$$1-\left(1-\frac{1}{m}\right)^{kn}.$$

Now test membership of an element that is not in the set. Each of the *k* array positions computed by the hash functions is 1 with a probability as above. The probability of all of them being 1, which would cause the algorithm to erroneously claim that the element is in the set, is often given as

$$\left(1-\left[1-\frac{1}{m}\right]^{kn}\right)^k \approx \left(1-e^{-kn/m}\right)^k.$$

This is not strictly correct as it assumes independence for the probabilities of each bit being set. However, assuming it is a close approximation we have that the probability of false positives decreases as *m* (the number of bits in the array) increases, and increases as *n* (the number of inserted elements) increases. For a given *m* and *n*, the value of *k* (the number of hash functions) that minimizes the probability is

$$\frac{m}{n}\ln 2 \approx 0.7\frac{m}{n},$$

which gives the false positive probability of

$$2^{-k} \approx 0.6185^{m/n}.$$

The required number of bits *m*, given *n* (the number of inserted elements) and a desired false positive probability *p* (and assuming the optimal value of *k* is used) can be computed by substituting the optimal value of *k* in the probability expression above:

$$p = \left(1 - e^{-(m/n \ln 2) n/m}\right)^{(m/n \ln 2)}$$

which can be simplified to:

$$\ln p = -\frac{m}{n} \left(\ln 2\right)^2 .$$

This results in:

$$m = -\frac{n \ln p}{(\ln 2)^2}.$$

This means that in order to maintain a fixed false positive probability, the length of a Bloom filter must grow linearly with the number of elements being filtered. While the above formula is asymptotic (i.e. applicable as m, n → ∞), the agreement with finite values of m,n is also quite good; the false positive probability for a finite bloom filter with m bits, n elements, and k hash functions is at most

$$\left(1 - e^{-k(n+0.5)/(m-1)}\right)^k .$$

So we can use the asymptotic formula if we pay a penalty for at most half an extra element and at most one fewer bit Goel & Gupta (2010) [22].

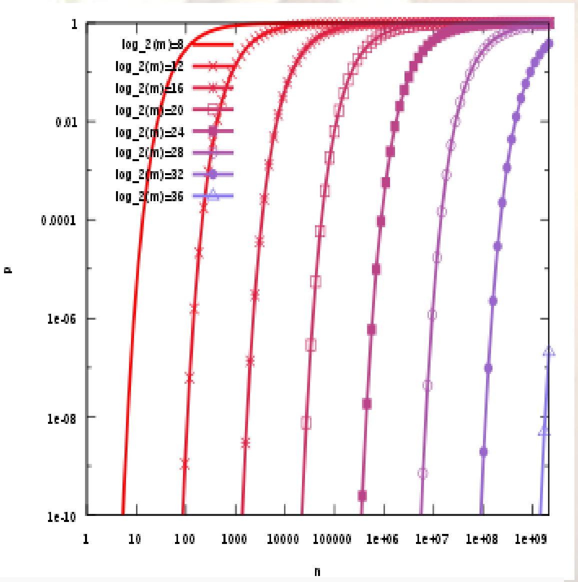

Fig.2: The false positive probability *p* as a function of number of elements *n*in the filter and the filter size *m*. An optimal number of hash functions$k = (m / n)\ln 2$ has been assumed.

IV. OPERATION ON BLOOM FILTERS

For the purposes of the analysis, we are presenting only the essentials of Bloom filters- the algorithms are for single bit elements. The analysis of filters with more complicated cells is essentially the same as for the simple case [23]. Blustein has shown how to efficiently implement these operations using C. IsMember is presented immediately below.

```
Procedure 1 (IsMember)
IsMember($Table$,$Key$)$\rightarrow$ Boolean
 1. $i \leftarrow 0$
 2. repeat
 3.     $i \leftarrow i + 1$
        $\triangleright$ $h_i$ is the $i^{th}$ hash transform, where $1 < i \leq m$
 4. until $((i = m) \vee \neg($IsSet$(Table[h_i(Key)])))$
 5. if  $i = m$  then
 6.     return(IsSet($Table[h_i(Key)]$))
 7. else
 8.     return(False)
end.
```

Fig. 3: Algorithm for Operation on Bloom Filter

V. SCOPE DECAY BLOOM FILTERS (SDBF)

A SDBF is designed as a lossy channel coding scheme to reduce the amount of network traffic. An SDBF can represent the set membership information and the different amount of information about an element in the set. Similar to a BF, an SDBF also has a bit string of width *m* and *d* hash functions, *h*1, *h*2, ..., and *hd*. An SDBF encodes the information about an element similarly to the way a BF inserts an element. Given an element *e*, the SDBF sets all bits *h*1(*e*), *h*2(*e*), ..., and *hd*(*e*) in the bit string. An SDBF differs from the basic BF in the decoding procedure. A BF obtains the membership information by checking whether all mapped *d* bits are set or not. An SDBF decodes the information about an element *e* by computing the number of 1s among the *d* mapped bits, denoted by *I*(*e*). This number ranges from 0 to *d*. The more bits are set to 1, the larger *I*(*e*) is, and there is more information about *e*.

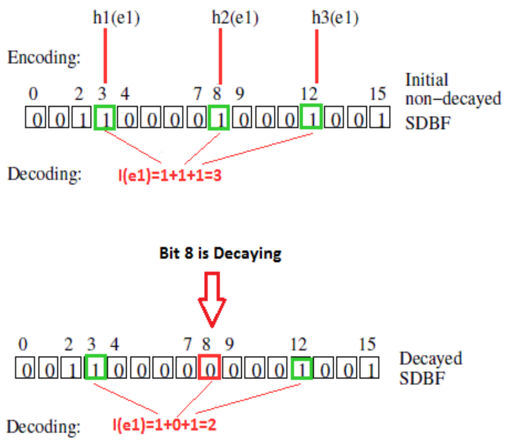

Fig.4: The structure of Scope Decay Bloom Filter (SDBF)

In this SDBF where, a bit string of width m=16 and hash functions d=3. There are 3 hash functions, they are h1, h2, and h3. When the SDBF initially encodes the information about element e1, the hash functions h1, h2, and h3 hash e1 to bits 3, 8, and 12 respectively and set these bits to 1s. Then decode the e1, and compute the number of 1s, that is I(e1)=3. During the decay process, some bits in the initial SDBF are probabilistically reset to 0s. In the decayed SDBF in Fig 4, bit 8 is reset to 0, bits 3 and 12 remain 1s. When we decode *e*1 from this decayed SDBF, *I*(*e*1) = 2, which means that this decayed SDBF probably has less information about *e*1 than the initial SDBF. There are many design choices for decreasing the information about an element in an SDBF. To simplify the decay process, we choose two stateless decay schemes that do not need to remember the specific event contributing to a particular bit. These two models are the exponential decay and *the linear* decay.

VI. Detecting and Correcting the Decayed Bit in the SDBF by Using Hamming Code

The Hamming code can be applied to data units of any length and uses the relationship between data and redundancy bits. In the Hamming code, each r bit is the parity bit for one combination of data bits, as shown below:

r1:	bits 1, 3, 5, 7, 9, 11
r2:	bits 2, 3, 6, 7, 10, 11
r4:	bits 4, 5, 6, 7
r8:	bits 8, 9, 10, 11

11	10	9	8	7	6	5	4	3	2	1
d	d	d	r8	d	d	d	r4	d	r2	r1

Fig.5: Position of redundancy bits in Hamming Code

Each data bit may be included in more than one calculation.

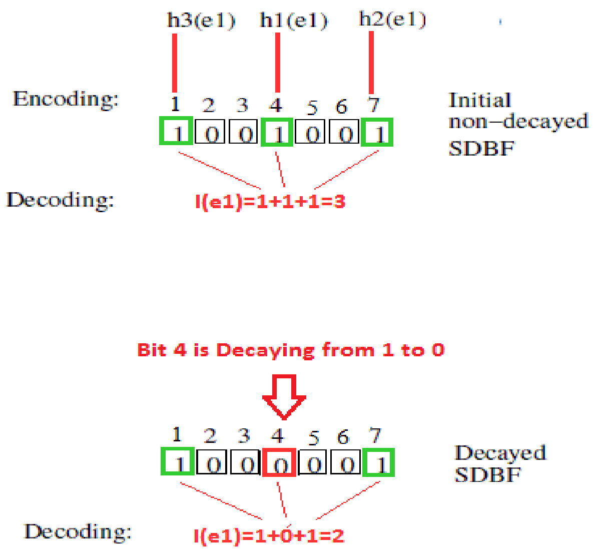

Fig.6: The initial non-decayed SDBF and decayed SDBF, the bit position 4 is decayed.

Calculating the r values: Fig.7 shows a Hamming code implementation for decayed SDBF of Fig.6. In the first step, we place each bit of the original character in its appropriate position in the 11-bit unit. In the subsequent steps, we calculate the even parities for the various bit combinations. The parity value for each combination is the value of the corresponding r bit.

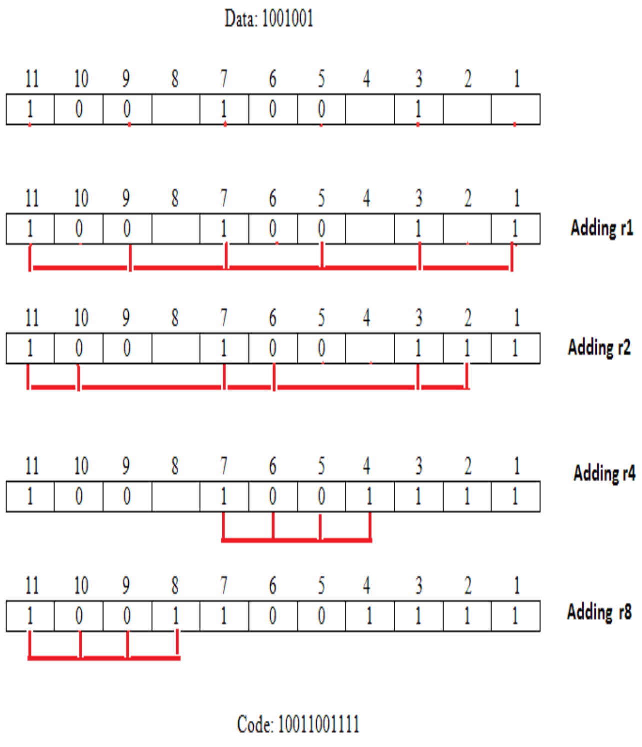

Fig.7: Redundancy bit calculation of Fig.5 SDBF

Error Detection and Correction: Now imagine that by the time the above transmission is received, the number 7 bit has been changed from 1 to 0. The receiver takes the transmission and recalculates 4 new parity bits, using the same sets of bits used by the sender plus the relevant parity r bit for each set. Then it assembles the new parity values into the binary number in order of r position (r8, r4, r2, r1). In our example, this steps gives us the binary number 0111 (7 in decimal (the bit in position 4 is decayed of Fig.6 SDBF)), which is the precise location of the bit in error

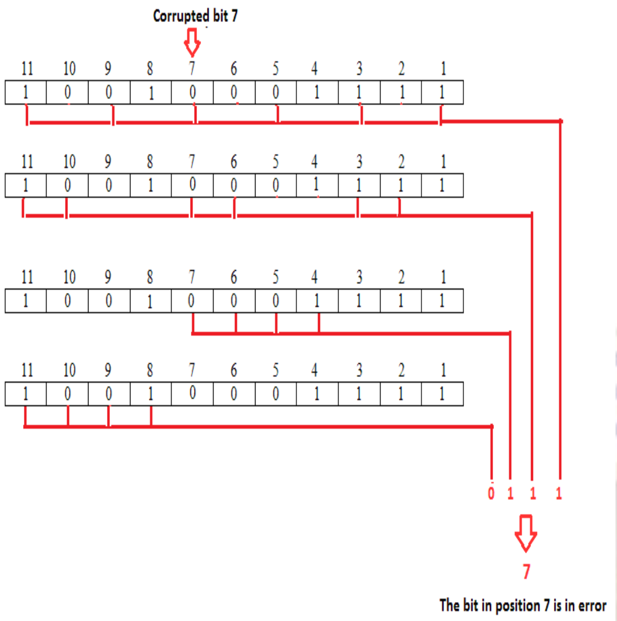

Fig.8: Error detection using Hamming code, the bit in position 7 (the bit in position 4 is decayed of Fig.6 SDBF) is in error.

VII. Probabilistic Routing Hints Creation and Maintenance

Probabilistic routing hints are represented by SDBFs. Each sensor maintains an SDBF for each neighbour. An SDBF encodes hints about events that may be found through a neighbour. To create these hints, each sensor first creates a local SDBF that encodes all local events detected by itself. Then these local SDBFs are propagated according to the decay model. At each sensor, the SDBF hints from different neighbours are first decayed if they contain information outside the *k*-hop neighbourhood of event sources, then aggregated (including the non-decayed local SDBF), and propagated further to other neighbours. A sensor first creates a local SDBF, encoding the set of events detected by itself. This SDBF is broadcast to all its neighbors. A neighbor combines this SDBF with the SDBFs from its own neighbours and propagates the aggregated SDBF. To reduce the routing traffic further, incremental updates to SDBFs are actually disseminated.

The figure shows how a node *X* propagates updates to its neighbors *Y*. *X ORs* its own SDBF and the SDBFs it receives from neighbour *A*, *B*, and *C* and sends the combined SDBF as hints to neighbour *Y*. If a sensor notices some change to its local SDBF, the changes are incrementally spread out to nearby nodes.

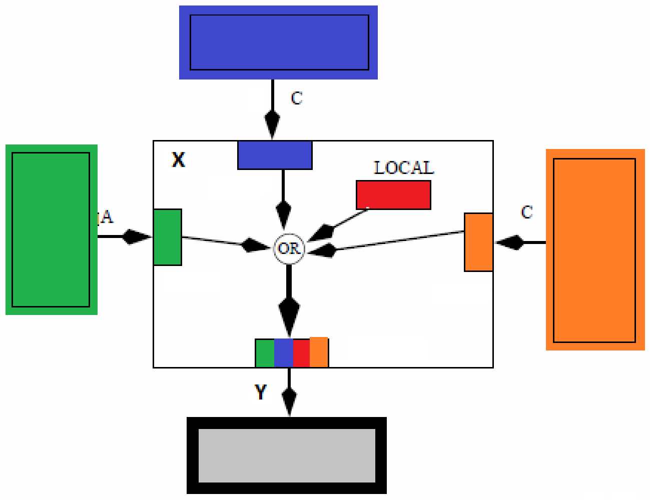

Fig. 9: Event hint update from *X* to its neighbour *Y*. A, *B*, *C* and *Y* are *X*'s neighbours. *LOCAL*: *X*'s local SDBF.

VIII. Conclusion

In this paper, we propose a routing protocol that detecting and correcting the decaying bits of SDBF using Hamming Code and utilize probabilistic hints. Each node maintains some probabilistic hints about events and utilizes these hints to route queries intelligently. The Bloom Filter is very important in WSNs. There are many important applications of BF in WSNs, such as Rule-based systems, Spell Checkers, Estimating Join Sizes, Differential Files, etc. SDBF greatly reduces the amortized network traffic without compromising the query success rate and achieves higher energy efficiency. SDBF generates less query traffic and achieves a significantly higher query success rate than SQR. In the future, we plan to explore scope decay bloom filters models, that decaying based on node degrees and intend to do analytical and simulation study in extending SDBF to clustered Wireless Sensor Networks (WSNs).

References

[1] Burton H. Bloom. Space/time trade-offs in hashing coding with allowable errors. *Communications of the ACM*, 13(7):422-426, July 1970. URL *h*http://doi.acm.org/10.1145/362686.362692*i*

[2] J. Hill, R. Szewczyk, A, Woo, S. Hollar, D. Culler, and K. Pister, System Architecture Directions for Networked Sensors, ASPLOS, November 2000.

[3] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “A survey on sensor networks,” *IEEE Communications Magazine*, pp. 102–114, Aug. 2002.

[4] D. P. Agrawal and Q. Zeng, *Wireless and mobile systems*. Thomson- Brooks/Cole, Inc., 2003.

[5] J. N. Al-Karaki and A. E. Kamal, "Routing Techniques in Wireless Sensor Networks: a Survey," *IEEE Wireless Communications*, vol. 11, no. 6, pp. 6–28, Dec. 2004.

[6] B. H. Bloom, "Space/time tradeoffs in hash codingwith allowable errors," *Communications of ACM*, vol. 13, no. 7, pp. 422–426, July 1970.

[7] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," in *Fortieth Annual Allerton Conference on Communication, Control, and Computing*, 2002.

[8] L. Fan, P.Cao, J. Almeida, and A. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," in *Proc. of ACM SIGCOMM'98*, Sept. 1998, pp. 254–265.

[9] A. Kumar, J. Xu, and E. W. Zegura, "Efficient and scalable query routing for unstructured peer-to-peer networks," in *Proc. of IEEE INFOCOM'05*, 2005.

[10] D. Guo, H. Chen, X. Luo, and J. Wu, "Theory and network application of dynamic bloom filters," in *Proc. of IEEE INFOCOM'06*, 2006.

[11] Burton H. Bloom. Space/time trade-offs in hashing coding with allowable errors. *Communications of the ACM*, 13(7):422-426, July 1970. URL

[12] James K. Mullin and Daniel J. Margoliash. A tale of three spelling checkers. *Software – Practice and Experience*, 20(6):625 - 630, June 1990.

[13] M. V. Ramakrishna. Practical performance of Bloom filters and parallel free-text searching. *Communications of the ACM*, 32(10):1237 { 1239, October 1989. URL *h*ttp://doi.acm.org/10.1145/67933.67941*i*.

[14] James K. Mullin. A second look at Bloom filters. *Communications of the ACM*, 26(8):570 – 571, August 1983. URL *h*http://doi.acm.org/10. 1145/358161.358167*i*.

[15] James K. Mullin. Optimal semijoins for distributed database systems. *IEEE Transactions on Software Engineering*, 16(5):558-560, May 1990. URL *h*ttp://ieeexplore.ieee.org/iel1/32/1900/0005 2778.pdf*i*.

[16] James K. Mullin. Estimating the size of a relational join. *Information Systems*, 18(3):189 – 196, 1993. ISSN 0306-4379.

[17] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: a Scalable and Robust Communication Paradigm for Sensor Networks," in *Proc. of ACM Mobi-Com*, 2000.

[18] C. Schurgers and M. B. Srivastava, "Engergy efficient routing in wireless sensor networks," in *Proc. of MILCOM communications for networkcentric ops: creating the information force*, 2001.

[19] R. C. Shah and J. Rabaey, "Energy aware routing for low energy ad hoc sensor networks," in *Proc. of IEEE WCNC*, 2002.

[20] L. Li, J. Halpern, and Z. Haas, "Gossip-based ad hoc routing," in *Proc. of the 21st Conference of the IEEE Communications Society (INFOCOM'02)*, 2002.

[21] Yu Hua, Bin Xiao, "A Multi-attribute Data Structure with Parallel Bloom Filters For Network Services"

[22] Goel, Ashish; Gupta, Pankaj (2010), "Small subset queries and bloom filters using ternary associative memories, with applications", *ACM Sigmetrics 2010* 38: 143, doi:10.1145/1811099.1811056

[23] Burton H. Bloom. Space/time trade-offs in hashing coding with allowable errors. *Communications of the ACM*, 13(7):422-426, July 1970. URL *h*ttp://doi.acm.org/10.1145/362686.362692*i*.