HR-SDBF: an approach to data-centric routing in WSNs

Xiuqi Li*

Department of Mathematics and Computer Science,
University of North Carolina at Pembroke,
P.O. Box 1510, Pembroke, NC 28272, USA
E-mail: xiuqi.li@uncp.edu
*Corresponding author

Jun Xu

College of Computing,
Georgia Institute of Technology,
801 Atlantic Drive, Atlanta, GA 30332-028, USA
E-mail: jx@cc.gatech.edu

Jie Wu

Department of Computer and Information Sciences,
Temple University, 324 Wachman Hall,
1805 N. Broad Street Philadelphia, PA 19122, USA
E-mail: jiewu@temple.edu

Abstract: In existing query-based routing protocols in wireless sensor networks (WSNs), a node either keeps precise route information to desired events, such as in event flooding, or does not keep any route to desired events, such as in query flooding. In this paper, we propose a routing protocol, called hint-based routing by scope decay bloom filter (HR-SDBF), that employs probabilistic hints. In the HR-SDBF protocol, each node maintains some probabilistic hints about the potential desired events and routes queries intelligently based on these probabilistic hints. We also put forward a data structure, scope decay bloom filter (SDBF) to encode the probabilistic hints. With SDBF, the amount of information about an event is propagated, without any loss, within the *k*-hop neighbourhood of the event source, but decreases outside the *k*-hop neighbourhood as the distance from the event source increases. Compared to existing query-based protocols, HR-SDBF greatly reduces the amortised network traffic without compromising the query success rate, and achieves a higher energy efficiency. To the best of our knowledge, this is the first query routing protocol in WSNs that utilises probabilistic hints encoded in a variant of the bloom filter. Both the analytic and the experimental results support the performance improvement of our protocol.

Keywords: bloom filters; BF; data-centric; hint-based; query-based; routing; wireless sensor networks; WSNs; high performance computing.

Reference to this paper should be made as follows: Li, X., Xu, J. and Wu, J. (2010) 'HR-SDBF: an approach to data-centric routing in WSNs', *Int. J. High Performance Computing and Networking*, Vol. 6, Nos. 3/4, pp.181–196.

Biographical notes: Xiuqi Li is an Assistant Professor at the Department of Mathematics and Computer Science in University of North Carolina at Pembroke, Pembroke, North Carolina, USA. She earned Faculty Summer Research Fellowship in 2010. She served as a Programme Committee Member and Session Chair in ten conferences. She holds 26 peer-reviewed journal and conference papers. Her research interests include networking, security, multimedia and web mining.

Jun Xu is an Associate Professor at College of Computing at Georgia Institute of Technology, Atlanta, USA. He earned the 2006 IBM Faculty Award, 2006 College of Computing Outstanding Junior Faculty Research Award, NSF CAREER Award and Ameritech Fellowship for outstanding research in telecommunications. He has published in several journals and refereed conference papers. His research interest is networking.

Jie Wu is a Chair and a Professor in the Department of Computer and Information Sciences, Temple University. Prior to joining Temple University, he was a Program Director at National Science Foundation. His research interests include wireless networks and mobile computing, routing protocols, fault-tolerant computing, and interconnection networks. He has published more than 500 papers in various journals and conference proceedings. He serves in the editorial board of the *IEEE Transactions on Computers*, *IEEE Transactions on Mobile Computing*, and *Journal of Parallel and Distributed Computing*. He is Program Co-Chair for IEEE INFOCOM 2011. He was also General Co-Chair for IEEE MASS 2006, IEEE IPDPS 2008, and DCOSS 2009. He has served as an IEEE Computer Society Distinguished Visitor and is the Chairman of the IEEE Technical Committee on Distributed Processing (TCDP). He is a Fellow of the IEEE.

1 Introduction

Recent advances in wireless communications and electronics have led to the rapid development of wireless sensor networks (WSNs). There is a wide range of applications of WSNs, such as in the health industry, military, warehouse, and home environment (Akyildiz et al., 2002a, 2002b). 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 (Agrawal and Zeng, 2003). In many WSN applications, routing is query-based. A data destination node, also called a *sink*, initiates a query for some desired data, which is forwarded towards the hosting sensors (Al-Karaki and Kamal, 2004). Sinks can be static or dynamic. In this paper, we are interested in the latter case, where any sensor could issue a query. We refer to the queried data as events. Because sensors have limited power, one of the major challenges in designing WSN routing protocols is energy efficiency. One way to achieve this is to reduce the total routing traffic (Akyildiz et al., 2002a, 2002b).

Existing query-based routing protocols can be classified into two types. The first type, called *query flooding based*, is blind forwarding and does not proactively maintain any hints. Queries are flooded over the WSNs. Query flooding can find desired events quickly but is also costly because many query messages are generated. This is evident when many events are frequently queried. The second type, called *event flooding based*, employs precise routing hints to route queries. The second type can reduce query messages at the expense of heavy routing overhead. Specifically, keeping precise routing hints for many events is expensive. This is because each node keeps a precise list of events that may be found through each neighbour. The cost to create and update this list is prohibitive when the list is large.

In this paper, we propose a routing protocol, called *hint-based routing by scope decay bloom filter (HR-SDBF)*, that utilises probabilistic hints. In HR-SDBF, each sensor maintains probabilistic hints about events that may be found through its neighbours. The hints are encoded using the proposed variant of the bloom filter (BF) (Bloom, 1970; Fan et al, 1998; Guo et al, 2006), called *scope decay bloom filter (SDBF)*.

A BF is a lossy compression of a set for supporting membership queries. It 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. BFs have been used in database applications, web caching, searching in peer-to-peer networks, and network measurements. Unlike BFs, an SDBF can denote different amounts of information about an element and represent probabilistic membership.

The HR-SDBF protocol uses SDBFs to advertise the routing hints about an event. The advertisement is designed such that the amount of hints do not decay within the k-hop neighbourhood of the event source, but decay 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, HR-SDBF achieves a higher query success rate with the same or less amortised routing overhead.

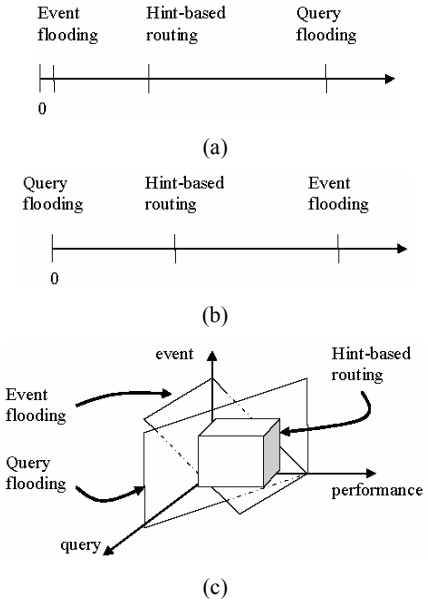
Sinks may conduct different types of searching based on SDBFs. They can specify the minimum amount of information that a neighbour must have in order to receive queries.

- *One-thread* best HR-SDBF. A query is always forwarded to the best neighbour that has the maximum amount of information among all neighbours. Ties can be broken by random selection or based on some SDBF component. This option is intended to find at least one desired event with the minimum cost.
- *N-threads* HR-SDBF. A query is forwarded to all neighbours that have the full amount of information about the desired event (i.e., all bits for the desired event are set). This choice is designed to find all events within the no-decay scope, k-neighbourhood.

Figure 1 illustrates the differences between the proposed hint-based scheme, query flooding and event flooding. Query cost refers to the number of query messages incurred per query. Routing overhead represents the cost for maintaining hints. Query flooding has the highest query cost

because it does not utilise any hints. Event flooding generates the minimum query traffic because each node keeps precise hints about all events. On the other hand, event flooding incurs the maximum routing over-head while query flooding does not have any routing over-head. The proposed hint-based scheme is in-between the two extremes.

Figure 1 Contrast hint-based routing with query flooding and event flooding (a) query cost (b) routing over-head (c) overall performance



Note: There are three dimensions: query, event, and performance.

We make the following contributions in this paper.

- We propose a hint-based routing protocol, HR-SDBF, which combines the advantages of both blind and precise-hint schemes. To the best of our knowledge, HR-SDBF is the first query routing protocol in WSNs that improves routing efficiency by utilising probabilistic hints.
- We present a novel data structure, SDBF. SDBFs improve the conventional BFs by being capable of representing probabilistic membership and various amount of information about elements. SDBFs are flexible and can include different decay models.
- We discuss different design tradeoffs in HR-SDBF. These include tie breaking by random selection and some SDBF component, and different decay models such as the exponential decay and the linear decay.
- We give some extensions of HR-SDBF, including extensions to clustered (or actor based) WSNs and applications of other BF variants.
- We conduct an extensive performance analysis and simulation of HR-SDBF.

This paper is organised as follows. In Section 2, we review existing query-based routing protocols in WSNs. In Section 3, we give an overview of the HR-SDBF protocol. In Section 4, we present the detailed design of HR-SDBF. We first introduce SDBF and different decay models, then describe how to create probabilistic hints and hint updates, and how to route queries using these hints. Some extensions of HR-SDBF are also discussed. In Section 5, we provide an analytical study of the HR-SDBF. In Section 6, we present experimental results about HR-SDBF's performance. In Section 7, we summarise the HR-SDBF protocol and point out the future work.

2 Related work

In this section, we review existing query-based routing schemes for flat WSNs that are associated with the work in this paper.

2.1 Query flooding and its variants

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 (Estrin et al., 1999; Intanagonwiwat et al., 2000) tries to find an optimal path between the sink and the event sources. A sink first initiates an exploratory query. Each node sets gradients between neighbouring nodes, and reinforces the best route for real databased on local rules while transferring the exploratory events. Note that the gradients are only used for sending the real data from the discovered event source to the sink that initiates the exploratory query. Directed diffusion also employs data caching and data aggregation to reduce network traffic.

Gradient-based routing (Schurgers and Srivastava, 2001) 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 neighbour. A gradient is defined as the height difference between a node and its neighbour. A node always forwards desired data through the link with the highest gradient among all links to its neighbours. Energy aware routing (Shah and Rabaey, 2002) 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 (Li et al., 2002) 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 neighbour.

2.2 Event flooding and its variants

Another option to correct the deficiency of query flooding is event flooding when the number of events in the WSN is small. We can use the minimum cost forwarding algorithm

in Ye et al. (2001) to set up the minimum cost path from every node to the event source. The event source broadcasts an event with cost 0. Each node updates its cost estimate and forwards a received message if the message leads to a lower cost path. To reduce the number of updates at some nodes, a backoff algorithm is applied at those nodes during the route setup.

Rumour routing (Braginsky and Estrin, 2001) combines query flooding and event flooding. It deploys an agent at a fixed probability for each event. An agent is a long-lived packet used to spread a list of events. An agent randomly walks in the WSN and synchronises its current event list with any node on its path. Synchronisation allows both agents and nodes to add new events and update routes to their existing events. There is only one route between an event source and a sink. After the route setup, when a sink queries for an event, if a neighbour has a route to that event, the query is forwarded to that neighbour along the known route. If no neighbour has a clue, the query is forwarded to a neighbour chosen uniformly at random. The neighbour repeats the same process. The query terminates when the TTL expires or when the desired event is reached.

Another related work is SQR searching (Kumar et al., 2005) in peer-to-peer networks that uses an exponential decay bloom filter (EDBF) to advertise hints. The major differences between SQR-EDBF and HR-SDBF are as follows. First, SDBF is more general and can incorporate different decay models. An EDBF can be considered as a one-hop SDBF with an exponential decay pattern. Second, SDBF can be utilised to perform more types of routing: one-thread and *N*-thread. The TTL counters embedded in SDBF can be employed for breaking ties. We can also specify the amount of hints for controlling the routing scope. SQR searching is a special case of HR-SDBF. It can be considered as the 1-thread HR-SDBF with random tie breaking.

To the best of our knowledge, only one existing query-based WSN routing protocol, called resilient data-centric storage (Ghose et al., 2003), uses BFs for routing. This protocol divides the sensor field into regions, each of which has a monitor node and at most one replica node for each event type. Unlike most WSN routing protocols where events are stored locally at the detecting sensors, this protocol stores information about all sources of an event type in some replica nodes. Each monitor node has a global view of all other monitor nodes and all replica nodes. A BF is used to represent the collection of attribute value pairs for all event types at each replica node. This protocol differs from our approach because the BF is not used for offering probabilistic hints.

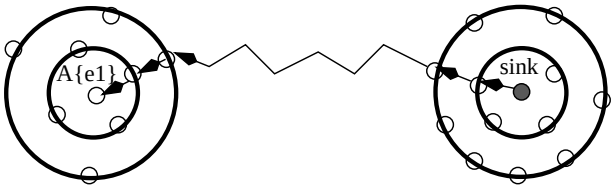
3 HR-SDBF protocol – overview

This section outlines the overall design of the HR-SDBF routing protocol. The design details will be discussed in the next section. The basic idea in the HR-SDBF protocol is to route the query based on probabilistic hints about the desired events. To obtain these probabilistic hints, we spread the knowledge about an event from the event source such that the amount of information about the event does not decay within the *k*-hop neighbourhood, but decreases outside the *k*-hop neighbourhood as the distance increases.

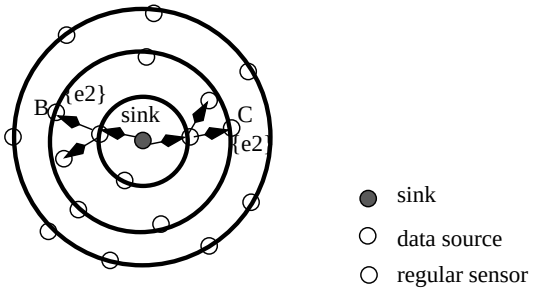
HR-SDBF protocol includes two types of searching, *one-thread best HR-SDBF* and *N-thread HR*-SDBF. The first approach is pictured in Figure 2(a). A query is always forwarded to the best neighbour that has the maximum amount of information among all neighbours. If multiple neighbours tie, we can select one randomly or break the tie based on some SDBF component. To control scope decay, each SDBF bit segment is equipped with a TTL counter. A counter for a segment is set to *k* at an event source if an event hashes to a bit in that segment. A TTL counter decreases by 1 at an advertisement. Ties can be broken by choosing the neighbour with the maximum TTL counter value associated with the desired event (i.e., selecting the one closest to an event source). A sink can control which neighbours are qualified for receiving queries by stipulating the minimum amount of information that a neighbour must have. This design choice is for finding at least one desired event efficiently.

The second approach is demonstrated in Figure 2(b). A query is redirected to all neighbours whose SDBF sets all bits of an event. A neighbour does not receive the query if it does not have the full amount of information. This scheme is targeted at efficiently locating all events within a no-decay *k*-neighbourhood.

Figure 2 HR-SDBF routing overview



(a) 1−thread best search



(b) N−thread HR−SDBF search

Notes: e_1, e_2: detected events. A, B, C, D: event sources.

The hint update is accomplished as follows. A sensor first creates a local SDBF, encoding the set of events detected by itself. This SDBF is broadcast to all its neighbours. A neighbour 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.

We consider two options for decreasing the information about an event, exponential decay and linear decay. In the exponential decay model, each bit that is currently 1 in an SDBF remains 1 at a constant probability p. The amount of information about an event at a node outside the k-hop neighbourhood is an exponential function of the distance from the boundary of the k-hop neighbourhood of the event source. In the linear decay model, the information about an event is approximately a linear function of the distance from the boundary of the k-hop neighbourhood. Neither decay model couples the amount of decay with a particular event source and a particular distance. Therefore, they do not need to memorise states.

4 HR-SDBF protocol – detailed design

This section presents the design details of the HR-SDBF routing protocol. We first introduce our SDBF BF. Then we discuss how to use the SDBF to build up and maintain the probabilistic routing hints. At the end, we describe how to route queries intelligently with the help of these probabilistic routing hints. We make the following assumptions about the WSN we are concerned about. The event source model is the random source model where event sources are selected uniformly at random from all sensors. Each sensor may potentially initiate event queries. An event query is considered successful if at least one desired event is found.

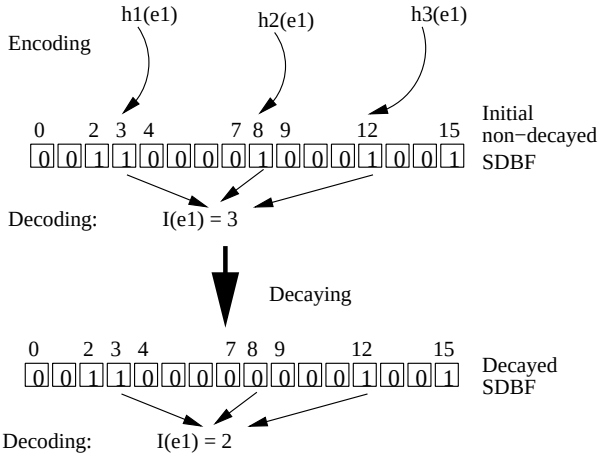
4.1 Scope decay bloom filter

An 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 h_d. 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 $h_d(e)$ in the bit string. An SDBF differs from a 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 set to 1, the larger $I(e)$, and the more information about e will exist.

Figure 3 shows an example of an SDBF where m = 16 and d = 3. The SDBF is composed of a 16-bit string *bstr* and three hash functions, h_1, h_2 and h_3. When the SDBF initially encodes the information about element e_1, h_1, h_2 and h_3 hash e_1 to bits 3, 8, and 12 respectively and set these bits to 1s. When we decode e_1 from this initial SDBF, we apply these three hash functions to e_1, and compute the number of 1s, $I(e_1)$, in bit positions 3, 8, and 12. Clearly $I(e_1)$ has the maximum value 3 that corresponds to the maximum information about e_1.

During the decay process, some bits in the initial SDBF are probabilistically reset to 0s. In the decayed SDBF in Figure 3, 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 has less information about e_1 than the initial SDBF.

Figure 3 The structure of 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.

- *The exponential decay model*. The information about an element (e.g., an event) decreases exponentially as the distance from the boundary of the k-hop neighbourhood of an element source (e.g., the sensor detecting an event) increases, assuming that there is no hash collision. Specifically, if any bit in an SDBF is currently 1, it remains 1 at a constant probability p during each decay. The number of 1s corresponding to an element (an event e) at an i-hop distance from the element source is

$$I(e) = d * \left(p^{i-k}\right).$$

- *The linear decay model.* The information about an element decreases in a linear fashion as the distance from the k-hop boundary of the element source increases. Let *sumI* denote the current total number of 1s in an SDBF and r be a random number in the range $[c_1, c_2]$, where c_1 and c_2 are system parameters. Randomly select r bits among the *sumI* bits and reset these bits to 0. The total number of 1s in the SDBF after the decay, denoted by *sumI'*, is

$$sumI' = sumI - r.$$

Assuming no hash collision, a bit currently at value 1 is reset to 0 with the probability p_l.

$$p_l = r/sumI .$$

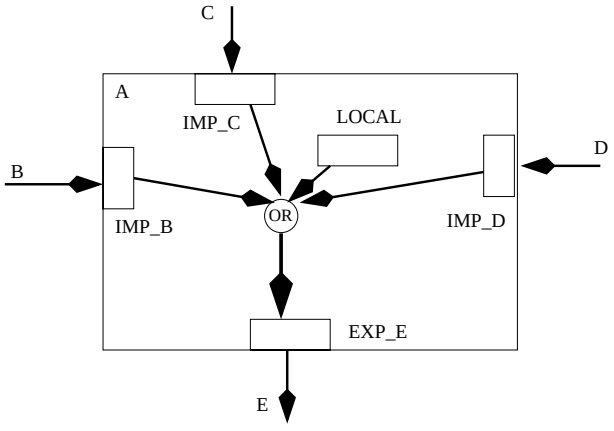
Obviously, a bit is more likely to be reset to 0 (i.e., decay faster) when the SDBF has a smaller number of 1s.

The SDBF can also be used to represent probabilistic sets. The number of 1s corresponding to an element can be considered as the probability of the element being in the set. The more 1s in the corresponding d bits, the more likely the element is in the set.

4.2 Probabilistic routing hint 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. Figure 4 shows how a node *A* propagates updates to its neighbour *E*. *A ORs* its own SDBF and the SDBFs it receives from neighbour *B*, *C* and *D*, and sends the combined SDBF as hints to neighbour *E*. If a sensor notices some change to its local SDBF, the changes are incrementally spread out to nearby nodes.

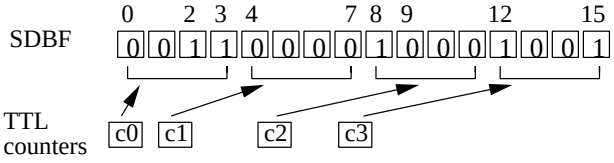
Figure 4 Event hint update from *A* to its neighbour *E*



Notes: *B*, *C*, *D* and *E* are *A*'s neighbours. LOCAL: *A*'s local SDBF. IMP_i: the SDBF hint imported (received) from neighbour *i*.

The control of no-decay within *k*-hops is illustrated in Figure 5. An SDBF is partitioned into *n* segments and one TTL counter is equipped for each SDBF segment. The counter for a segment has the same value as the decay range *k* at an advertising source that contributes to the segment. In the example, the SDBF is 16-bit and the segment size is 4 bits. Only 4 TTL counters are required.

Figure 5 Control no decay within *k*-hop neighbourhood of an event source



1 TTL counter per SDBF segment
(segment size = 4 bits)

Figure 6 and Figure 7 show how to create a local SDBF and hint update at a sensor *s*. To create a local SDBF LR_s for the local event set LE_s, *s* first checks whether LE_s is empty. If not, *s* hashes each event e using d hash functions and sets the corresponding d bits in LR_s to 1s. *s* also initialises the TTL counter for each segment to the maximum value *k*. If LE_s is empty, LR_s has all 0s in its bit array and counter array. To create a hint update UR_{n_i} to a neighbour n_i, *s* first assigns LR_s to UR_{n_i} if LE_s is not empty. Then for each segment in the SDBF of each neighbour n_j other than n_i, *s* calls the procedure *Decay*() to process the SDBF segment according to the pre-defined decay model.

Figure 6 Algorithms for creating local hints and hint updates

```
Create a local SDBF LR_s for the local event set LE_s:
    // Assume: one TTL counter per SDBF segment.
    // g: the segment size
    // LR_s.barr: the bit array for encoded elements
    // LR_s.carr: the TTL counter array
1.  if !empty(LE_s)
2.      Reset LR_s;
3.      ∀e ∈ LE_s
4.          Set bits LR_s.barr[h_1(e)], ..., LR[h_d(e)] to 1;
5.      ∀i ∈ {1, ··· , m/g};
6.          if Count Ones (LR_s.barr, i) > 0;
7.              LR_s.carr[i] = k;
8.          else
9.              LR_s.carr[i] = 0;

Create a hint update UR_{n_i} to neighbor n_i:
1.  if !empty(LE_s)
    // Include the non-decayed local SDBF LR_s
2.      UR_{n_i} = LR_s;
3.  else
4.      Reset UR_{n_i}.barr and UR_{n_i}.carr;
    // Process each segment in each SDBF from a neighbor.
5.  ∀n_j ∈ my_neighbors such that n_i ≠ n_j
6.      ∀i ∈ {1, ··· , m/g};
7.          if Count Ones (UR_{n_j}.barr, i) > 0;
8.              if UR_{n_j}.carr[i] - 1 > UR_{n_i}.carr[i]
9.                  UR_{n_i}.carr[i] = UR_{n_j}.carr[i] - 1;
10.             if UR_{n_j}.carr[i] == 0
11.                 seg = extractSeg(UR_{n_j}.barr, i);
12.                 seg = Decay(seg);
13.         UR_{n_i}.barr = OR(UR_{n_i}.barr, i, seg);
14. Return UR_{n_i};
```

Figure 7 Algorithm for decay an SDBF segment

```
Decay (R)
   // Decay the SDBF segment R
   // according to the predefined decay model.
   // R_d: the segment after decaying R.
1. Reset R_d;
2. if exponential decay
3.      ∀i ∈ {1,···,g}
4.          if R[i] == 1
5.              Set R_d[i] to 1 with probability p;
6.      Return R_d;
   // linear decay model
7. R_d = R;
8. sumI = Count Ones ( R );
9.      Pick a random number r ∈ [c1, c2];
10.     ∀i ∈ {1,···,g}
11.         if R[i] == 1
12.             Reset R_d[i] with probability r/sumI;
13. Return R_d;
```

4.3 Query routing based on SDBF hints

Figure 8 illustrates the one-thread best search in HR-SDBF. The sink finds all neighbours whose SDBFs have at least *minBits* set among the bits for event *e*. These qualified neighbours are ranked according to the number of bits set for *e*. The query is routed to the top-ranked neighbour. On receiving Q_e, a neighbour checks if the query is a duplicate. If so, Q_e is dispatched to a random neighbour. If *e* is a local event at this neighbour, the query forwarding terminates. If not, Q_e is redirected similarly if the query TTL does not expire.

During query forwarding, there may be top-one ties. If tie breaking is random, a neighbour among all ties is randomly selected as the best neighbour. If the tie breaking is TTL counter, a max-min strategy is used to select the best neighbour. First, for each neighbour, the minimum TTL counter value among those of all segments in association with *e* is chosen as the TTL counter value for that neighbour. Then the neighbour with the maximum TTL counter value with respect to e is considered the best.

Figure 8 One-thread best HR-SDBF search

```
1-thread best search: process Q_e at sink s:
1. getBestNbr(minBits, bestNbr, s, Q_e);
2. if bestNbr != NULL
3.      −−Q_e.TTL;
4.      Deliver Q_e to bestNbr;

1-thread best search: process Q_e at t that is not a
sink:
1. if pastQuery ( Q_e )
2.      if Q_e.TTL > 0
3.          Deliver Q_e to a random neighbor n_i;
4.      Return;
5. if e ∈ LE_s
6.      Q_e.found = true;
7.      Send e back to the sink;
8.      Return;
9. getBestNbr(minBits, bestNbr, s, Q_e);
10. if Q_e.TTL > 0 and bestNbr != NULL
11.     −−Q_e.TTL;
12.     Deliver Q_e to bestNbr;
```

Figure 9 Locate the best neighbour for one-thread HR-SDBF

```
getBestNbr(minBits, bestNbr, s, Q_e):
1. Find all neighbors, qualNbrs , with I_{n_i}(e) ≥ minBits;
2. if qualNbrs.size > 0
3.      Sort qualNbrs in decreasing I_{n_i}(e);
4.      Compute topOneTies;
5. if topOneTies > 1
6.      if randomTieBreak
7.          Pick a random number r_i in [0, topOneTies − 1];
8.          bestNbr = qualNbrs[r_i];
9.      else
10.         ComputeAvgTTLCounts(e,topOneTies,qualNbrs);
11.         Find r_i with maximum qualNbrs[r_i].avgTTLCount;
12.         bestNbr = qualNbrs[r_i];
```

The rationale for the max-min TTL counter strategy is explained as follows. When an SDBF segment is only used by one event, the TTL counter for that segment is set to *k* at the event source and decreased by one (until 0) during each advertisement. When two events share a segment, if the TTL counter for the segment is set according to one event, the other event can only cause the shared TTL counter to stay the same or increase but not decrease. Therefore, in each neighbour's SDBF, among all segments of an event, if at least one segment's TTL counter value is not changed by other events, the minimum TTL value is the correct value for that event. If TTL counters of all segments that are related to an event are false, the minimum TTL counter value has the smallest error.

If all neighbours' TTL counter values are correct with respect to an event, the neighbour with the maximum TTL counter value is closest to the event source. Otherwise, there is no way to distinguish between true and false TTL counter values. Choosing the neighbour with the maximum TTL counter value is like a random selection.

In the *N*-thread search, the *minBits* must be the same as *d* and the maximum query TTL must equal to *k*. The sink forwards a query Q_e to all neighbours with all d bits set for event *e*. When a neighbour receives a non-duplicate query Q_e for a local event *e*, the detailed information about *e* is returned to the sink. The neighbour continues to send Q_e similarly to the sink until the query TTL expires. Duplicate queries are discarded at the receiving neighbour.

4.4 Extensions

The HR-SDBF routing can be extended in the following ways. First, we can run HR-SDBF on a hierarchical topology to increase scalability. The WSN can be divided into non-overlapping clusters with one clusterhead (CH) per cluster. Clusters can be formed by data correlation. Each CH computes a local SDBF for events detected within its cluster. Each CH also aggregates and propagates SDBF hints from other neighbour CHs. When a CH receives a query, it first checks its own cluster. If the query is not resolved, the CH forwards the query to the best neighbour CH that has the most hints about the desired event. HR-SDBF can also be extended to actor-based WSNs (Akyildiz and Kasimoglu, 2004; Melodia et al., 2005) in a similar way. Second, we can combine compressed BFs

(Mitzenmacher, 2002) and SDBFs. CHs or actors compress an SDBF using arithmetic coding before it is propagated.

5 Analysis

In this section, we analyse the query traffic of the HR-SDBF routing protocol. The analysis focuses on one-thread forwarding with the exponential decay model. We prove the best query performance within no-decay scope and the average query traffic outside the no-decay scope. The simulation study of the HR-SDBF protocol will be presented in the next section.

We assume that sensors are randomly deployed. The network topology is a random graph where each neighbour has the same average degree $n_c = N_{sn} * \pi * R^2 / TD_s$, where N_{sn} refers to the number of sensors in the WSN; R is the wireless transmission radius of a sensor; TD_s is the total area of the sensing field.

Theorem 1 (Best query performance within k-hops): In the best scenario, the one-thread query forwarding within k-hops of an event source follows the shortest path if the max-min TTL counter strategy is used to break ties.

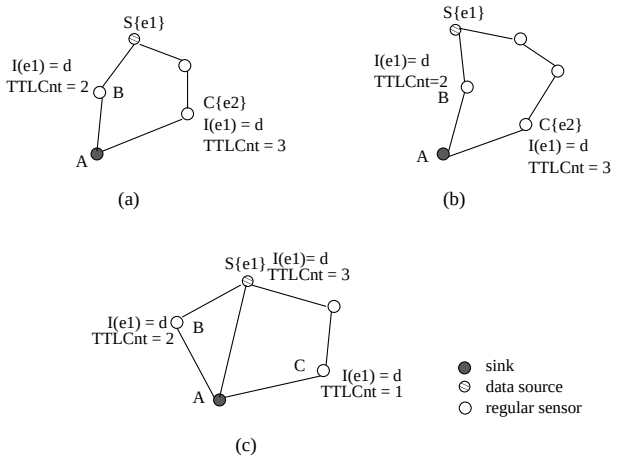
Proof: Within k-hops of an event source, a query may be forwarded to a neighbour that is not actually the best due to two factors, sharing one TTL counter ($g > 1$) and hash collision. When $g > 1$, as shown in Figure 10(a), A is the query source, S has the desired event e_1. At node C, a local event e_2 hashes to a different bit in each segment of event e_1. This causes the TTL counter value for C with respect to e_1 to be 3. The max-min TTL counter strategy incorrectly chooses C as the best neighbour. When $g = 1$, hash collision may still cause a false selection of the best neighbour. For example in Figure 10(b), a local event e_2 at node C hashes to the same set of bits as the desired event e_1, which causes $I(e_1)$ at C to be d and the TTL counter value for C with respect to e_1 to be 3. C is incorrectly selected by the max-min TTL counter strategy to be the best neighbour.

There are four scenarios in which the two factors together cause a false selection of the best neighbour.

- $g = 1$ and no hash collision
- $g = 1$ and hash collision
- $g > 1$ and no hash collision
- $g > 1$ and hash collision.

The best scenario is the first one, where there is one TTL counter per bit and no two events ever hash to the same bit. As illustrated in Figure 10(c), the TTL counter values for all neighbours are true values for the desired event. And they represent the shortest distance between neighbours and the event source. Therefore, the best one-thread query forwarding within k-hops follows the shortest path. □

Figure 10 (a) Error due to shared TTL counters ($g > 1$) alone (b) error due to hash collision ($g = 1$) alone (c) best scenario in one-thread forwarding



Proposition 1 (Impact of false positives): The number of bits set due to false positives in an SDBF-ED follows a binomial distribution $B(d, \lambda_p)$, where

$$\lambda_p = \left(1 - e^{-U/(n_c * m)}\right),$$

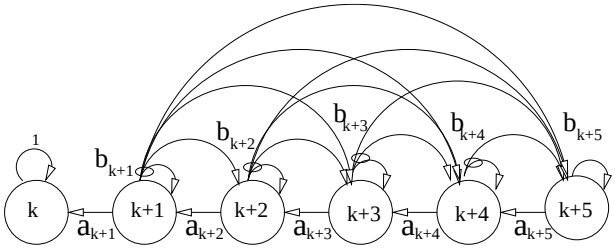
$$U \le \sum_{i=1}^{k} w_i * d + \sum_{i=k+1}^{j} w_i * d * p^{i-k} + d * p^{i-j} * \left(N_{sn} - \sum_{t=1}^{j} w_t\right).$$

w_i (w_t) denotes the number of nodes i (t) hops away from a sensor; k is the no-decay neighbourhood radius; d is the number of hash functions in an SDBF-ED; p is the decay rate; U represents the total number of bits that are caused by an event and propagated throughout the network.

Proof: See Appendix. □

The average energy consumed by a query is proportional to the average number of messages forwarded per query, N_{qs}, which is computed by modelling the query forwarding process as a Markov chain, as shown in Figure 11. There is one Markov chain for each query for events that actually exist in the WSN. Each sensor stays in exactly one state in each Markov chain. A sensor stays in state i, except k and $k + 5$, if its shortest distance from an event source is i. State k refers to nodes that are within a no-decay k-hop neighbourhood. State $k + 5$ refers to nodes that receive a negligible amount of information about a desired event. It is assumed that the hint about an event decays to a negligible amount outside the $k + 5$ neighbourhood. This assumption holds as long as $d * p^5 \ll 1$. Therefore all nodes outside the $(k + 5)$ – hop neighbourhood stay in state $k + 5$. The number of nodes in state i, denoted by ws_i, is the same as w_i mentioned above if $i \neq$ k; i $\neq k + 5$. $ws_k = \sum_{t=1}^{k} w_t$ and $ws_{k+5} = N_{sn} - \sum_{t=1}^{k+4} w_t$.

Figure 11 A Markov chain model of HR-SDBF-ED



Proposition 2 (state transition matrix C): The state transition probability matrix for the Markov chain of an event is the matrix C in Figure 12. a_i refers to the probability of transition from state i to state $i - 1$ when $i \in [k + 1, k + 5]$. b_{ij} denotes the transition probability from state i to state $j \geq i$ when $i \in [k + 1, k + 5]$.

$$a_i = P\left[\left(Z > F\right)^{n_c-1}\right] + \sum_{u=1}^{n_c-1} P\left[\left(Z = F\right)^{u}\left(Z > F\right)^{n_c-u-1}\right].$$

Z is the number of bits set to 1 among the d bits of an SDBF-ED that is received from the upstream neighbour on the shortest path of a sensor in state i to an event source. Z is a random variable that follows the binomial distribution $B(k, p^{i-k} + \lambda_p - \lambda_p * p^{i-k})$. $b_{ij} = \left(ws_i \Big/ \sum_{t=k+1}^{k+5} ws_t\right)\left(1 - a_i\right)$.

Figure 12 The state transition matrix C

1	0	0	0	0	0
a_{k+1}	$b_{(k+1)(k+1)}$	$b_{(k+1)(k+2)}$	$b_{(k+1)(k+3)}$	$b_{(k+1)(k+4)}$	$b_{(k+1)(k+5)}$
0	a_{k+2}	$b_{(k+2)(k+2)}$	$b_{(k+2)(k+3)}$	$b_{(k+2)(k+4)}$	$b_{(k+2)(k+5)}$
0	0	a_{k+3}	$b_{(k+3)(k+3)}$	$b_{(k+3)(k+4)}$	$b_{(k+3)(k+5)}$
0	0	0	a_{k+4}	$b_{(k+4)(k+4)}$	$b_{(k+4)(k+5)}$
0	0	0	0	a_{k+5}	$b_{(k+5)(k+5)}$

Proof: See Appendix. □

Proposition 3 (average query traffic outside k-hops): In the HR-SDBF-ED protocol, the average number of query messages forwarded by sensors outside k-hop, denoted by N_{qs}, is as follows.

$$N_{qs} = G * \left(I - G\right)^{-1} * \left(I - G\right)^{-1}$$

where G is the resulting matrix by changing the top-left element in C from 1 to 0.

Proof: Let $T_{i,0}$, $i = k + 1, k + 2, \ldots, k + 5$, denote the number of steps it takes for a query that starts at a node in state i to first encounter a node at distance k from the target (i.e., state k). Let T denote the number of steps for a query that starts at a node chosen uniformly at random to first encounter a node at distance k from the event source. Since the probability of a query originating in state i is ws_i/N_{sn}, therefore

$$E\left[T\right] = \sum_{i=k+1}^{k+5} \frac{ws_i}{N_{sn}} E\left[T_{i,0}\right]. \quad (1)$$

Let G be the matrix resulting from just changing the top-left element in C from 1 to 0. It is not difficult to verify, from the meaning of the stochastic matrix G, that the element in the ith row 0th column of G^l is the probability that when a node starts in state i, it first reaches a node at distance k from the event source after l steps. Therefore, the element in the ith-row 0th-column of the matrix $\sum_{j=1}^{\infty} j * G^j$ equals to $E\left[T_{i,0}\right]$. Based on the theory of polynomial series, $\sum_{j=1}^{\infty} j * Y^j = Y * \left(1 - Y\right)^{-2}$ for any complex variable Y within the convergence radius $\left|Y\right| < 1$. Similarly $\sum_{j=1}^{\infty} j * G^j = G * \left(I - G\right)^{-1} * \left(I - G\right)^{-1}$ after some convergence arguments for matrices. □

6 Evaluations

In this section, we will discuss the experimental setup, evaluate the tradeoffs in the HR-SDBF design, and compare HR-SDBF to SQR routing and query flooding.

6.1 Experimental setup

The sensor network is generated by randomly deploying 500 sensors in the field of size 200 m × 200 m. It is assumed that each node can reliably send packets to any node within R meters. R is set to 15 or 20. The event model is the random source. 250/1000/2500 events are randomly distributed among all sensors to simulate small/medium/large event scenarios. Each unique event has five replicas. Queries are generated by randomly selecting a sensor as a sink and an existing event as the desired one. In each simulation run for HR-SDBF, hints about events are first propagated according to the scope decay model, then queries are processed. The performance metrics are routing energy efficiency and routing quality. We assume that it costs more to set up a connection than to transmit a single message. The routing energy efficiency is computed in terms of the average number of query messages and the average number of amortised messages. The latter is defined as the sum of the total number of query messages and the routing update messages divided by the number of queries. The routing quality is evaluated based on the query success rate. A query is considered successful if at least one desired event is found. Table 1 lists the major system parameters.

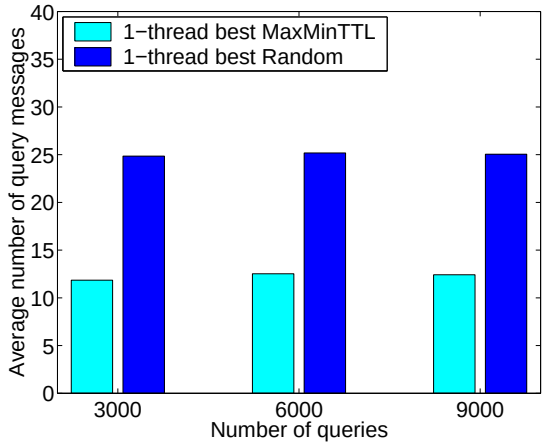
Table 1 Major system parameters

Notation	*Definition*	*Value*
m	The width of SDBF filter	12 kbits
d	The number of hash functions	16
g	The SDBF segment size	8
k	The no-decay scope	3
p	The exponential decay rate	1/8
($c1$, $c2$)	The linear decay control range	(3, 6)

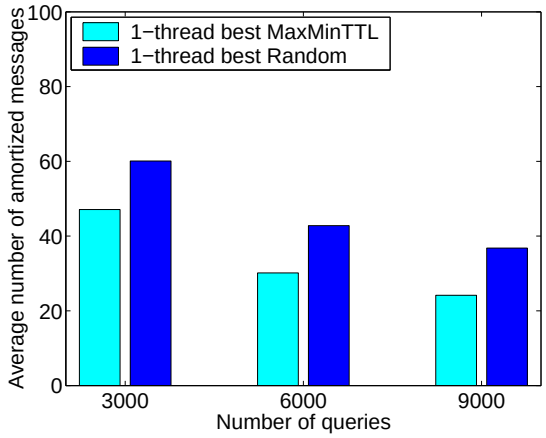
6.2 One-thread HR-SDBF: random vs. TTL tie-breaking

In the one-thread HR-SDBF protocol, a query is always forwarded to the best neighbour. Ties can be broken randomly or based on the max-min TTL counter strategy. Figure 13 contrasts the performance of these two design choices when the same scope decay model is used and different number of queries are executed. The number of events is 2500. The labels *one-thread best MaxMinTTLCnt* and *one-thread best random* refer to breaking ties using TTL counters and random tie-breaking respectively. Clearly, tie-breaking using TTL counter value surpasses random tie-breaking. It achieves a higher query success rate with less query messages and less amortised messages. This suggests that in HR-SDBF we can use both the BF and the TTL counter as hints about event locations. The TTL counter tie-breaking is a better design choice.

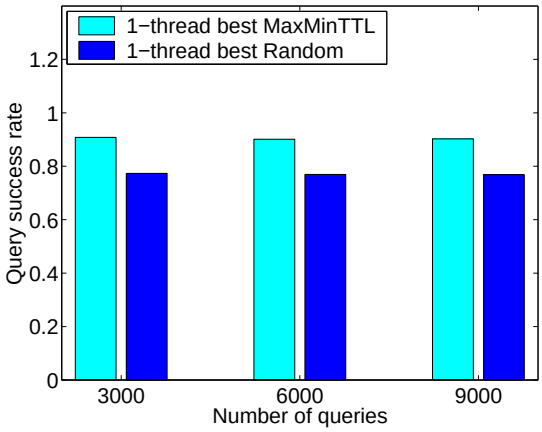
Figure 13 One thread HR-SDBF: random tie breaking vs. TTL tie-breaking (a) average number of query messages (b) average number of amortised messages (c) query success rate (see online version for colours)



(a)

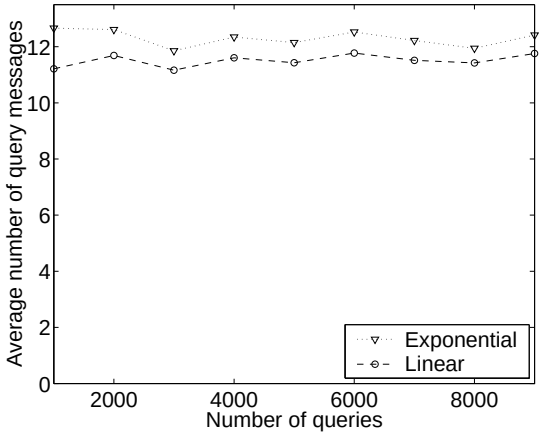


(b)

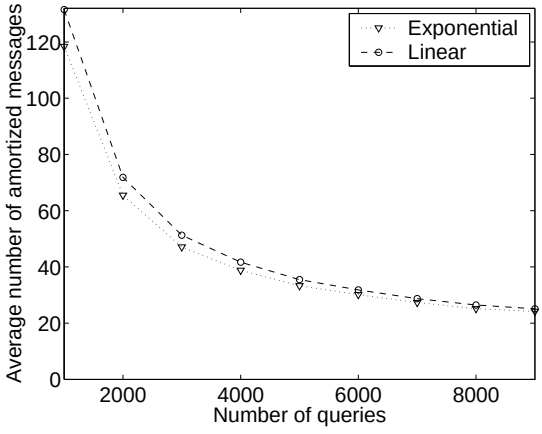


(c)

Figure 14 One-thread HR-SDBF: exponential decay vs. linear decay (a) average number of query messages (b) average number of amortised messages (c) query success rate

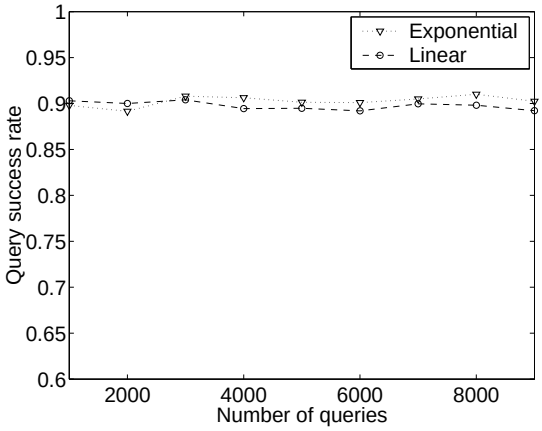


(a)



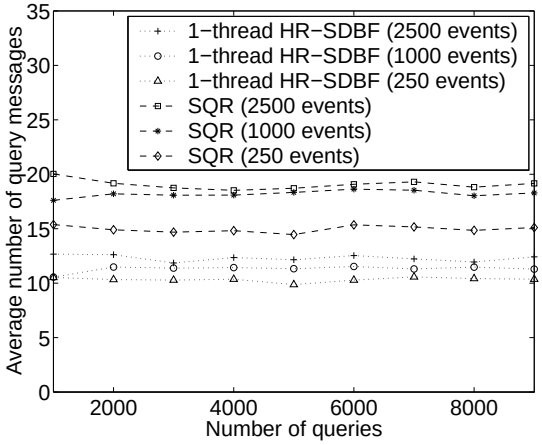
(b)

Figure 14 One-thread HR-SDBF: exponential decay vs. linear decay (a) average number of query messages (b) average number of amortised messages (c) query success rate (continued)

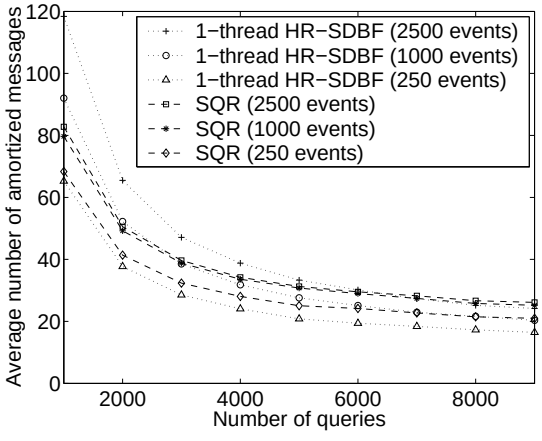


(c)

Figure 15 One-thread HR-SDBF compared to SQR, varying number of queries (R = 15) (a) average number of query messages (b) average number of amortised messages (c) query success rate

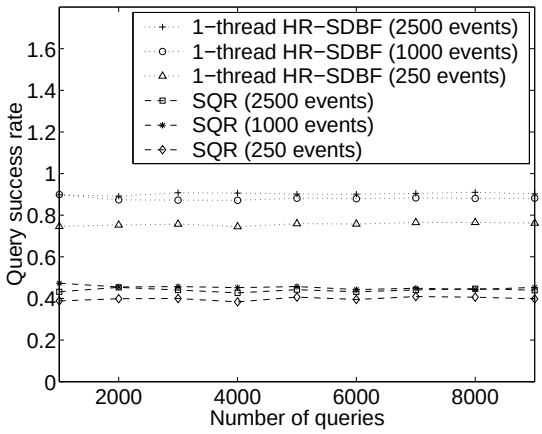


(a)



(b)

Figure 15 One-thread HR-SDBF compared to SQR, varying number of queries (R = 15) (a) average number of query messages (b) average number of amortised messages (c) query success rate (continued)



(c)

6.3 Decay models: exponential vs. linear

Another tradeoff in the HR-SDBF design is exponential decay or linear decay. Figure 14 shows the performance contrast between these two choices. The number of events is 2,500. The search type is one-thread HR-SDBF with tie breaking by max-min TTL counter policy. Figure 14(a) indicates that the two decay models generate approximately the same amount of query traffic. Exponential decay incurs less routing overhead than linear decay, as shown in Figure 14(b). In addition, slightly more queries are successfully resolved with exponential decay than with linear decay, as illustrated in Figure 14(c). Therefore, exponential decay is slightly better than linear decay.

6.4 One-thread HR-SDBF vs. SQR

The one-thread HR-SDBF with tie-breaking using max-min TTL counter values is compared to SQR routing. We first compare their performance by varying the number of queries (i.e., varying the frequency of an event being queried) and the number of events. This is shown in Figure 15.

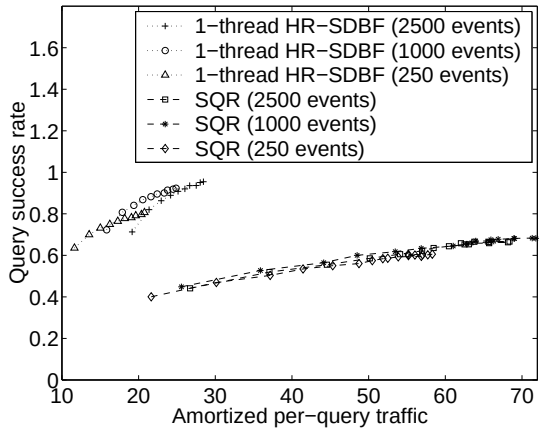
Each line in Figure 15(a) plots the average number of query messages per query when the number of queries increases and the maximum TTL for a query is fixed at 50. Different lines correspond to the performances of different algorithms under three different event scenarios: 2,500 events, 1,000 events, and 250 events. Each unique event has five replicas. The figure indicates that in the same event scenario, the average query traffic in neither scheme increases as events are queried more frequently. One-thread HR-SDBF incurs about 33% less query traffic than SQR in all three event scenarios. This is because HR-SDBF does not decay event hints within k-hop neighbourhoods of event sources and therefore can propagate hints further.

The amortised messages including the update overhead is illustrated in Figure 15(b). In the 250-event scenario, one-thread HR-SDBF always has lower amortised traffic than SQR. This is because HR-SDBF generates about the same amount of hint propagation traffic as SQR but significantly less query traffic than SQR. In the other event scenarios, both schemes generate decreasing amortised traffic with increasing queries. One-thread HR-SDBF has higher amortised traffic than SQR when the number of queries is small. However, the difference decreases dramatically as the number of queries increases. In the 1,000-event/2,500-event scenario, HR-SDBF generates less amortised traffic than SQR when the number of queries is greater than 3,000/6,000. This means that the extra hint propagation traffic caused by no-decay within *k*-hops in HR-SDBF is effectively amortised by the increase in query frequency.

Figure 15(c) shows the query success rates of both schemes when the number of queries increases. It is observed that in the same event scenario, the number of queries does not impact the query success rate in either scheme. One-thread HR-SDBF achieves a dramatically higher query success rate than SQR in all three event scenarios. The query success rate of HR-SDBF is about twice as much as that of SQR in the same event scenario. This significant increase is due to the fact that HR-SDBF can push event hints further than SQR.

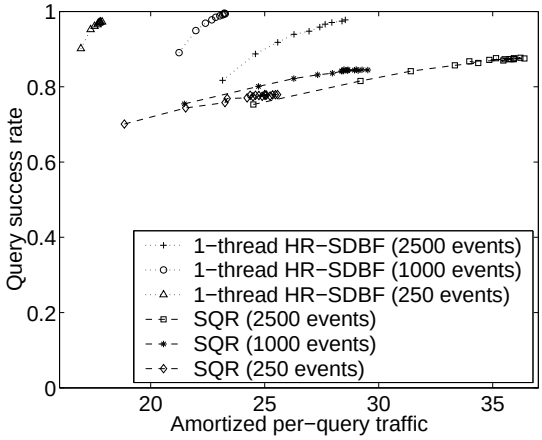
To compare the performance of 1-thread HR-SDBF and SQR when they incur the same per-query traffic, we gathered data with varying maximum query TTLs and plot the query success rate in terms of the amortised per-query traffic, as shown in Figure 16. The number of queries is 8000. Clearly, 1-thread HR-SDBF is superior to SQR. It can achieve a higher query success rate than SQR with the same amount of amortised traffic in all three event scenarios.

Figure 16 One-thread HR-SDBF vs. SQR, query success rate in terms of amortised per-query traffic (a) $R = 15$ (b) $R = 20$



(a)

Figure 16 One-thread HR-SDBF vs. SQR, query success rate in terms of amortised per-query traffic (a) $R = 15$ (b) $R = 20$ (continued)

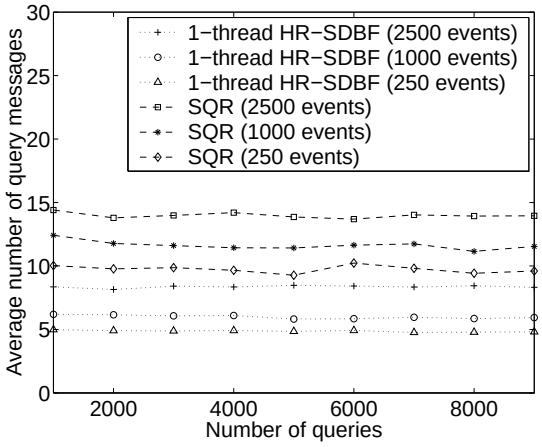


(b)

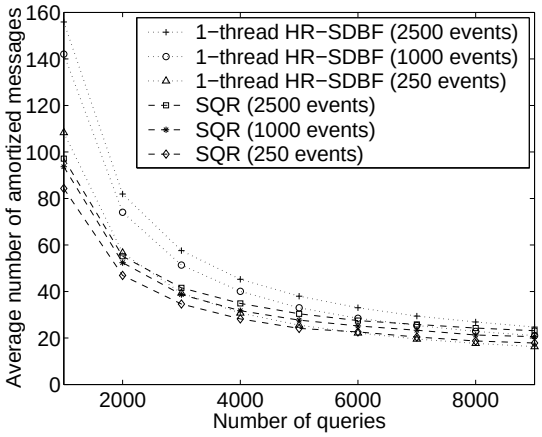
We also simulate one-thread HR-SDBF and SQR in denser networks where the wireless communication radius is increased from 15 meters to 20 meters. All other system parameters remain the same. Their performance with respect to varying number of queries and different event scenarios are pictured in Figure 17. Similar trends are observed in denser networks, but the absolute values are different. In denser networks, the average query traffic is lower, the amortised per-query traffic is slightly heavier, and the query success rate is higher. This is because in denser networks, nodes have more neighbours and the network diameter is smaller. Compared to SQR, HR-SDBF generates 40% less query traffic, and increases the query success rate around 28%. The amortised traffic is only 8% more than SQR when the number of queries is more than 6,000. Figure 16(b) shows the changes of the query success rate with respect to the amortised traffic in the denser network when the number of queries is fixed at 8,000 and the query TTL changes. The one-thread HR-SDBF still outperforms SQR though SQR performs much better in densor networks.

In summary, HR-SDBF generates less query traffic and achieves a significantly higher query success rate than SQR. This is because HR-SDBF does not decay event hints within *k*-hop neighbourhoods of event sources. Therefore, hints can propagate further. When there are many events, this no-decay within *k*-hops also causes more traffic in spreading hints. But the decrease in the query traffic out-weighs the increase in the hint maintenance traffic at high query frequencies.

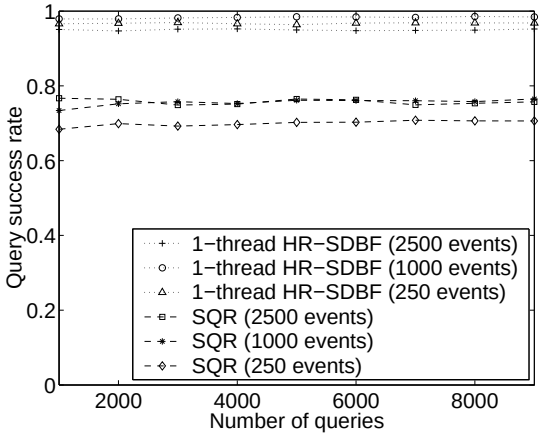
Figure 17 One-thread HR-SDBF compared to SQR, varying number of queries (R = 20) (a) average number of query messages (b) average number of amortised messages (c) query success rate



(a)



(b)



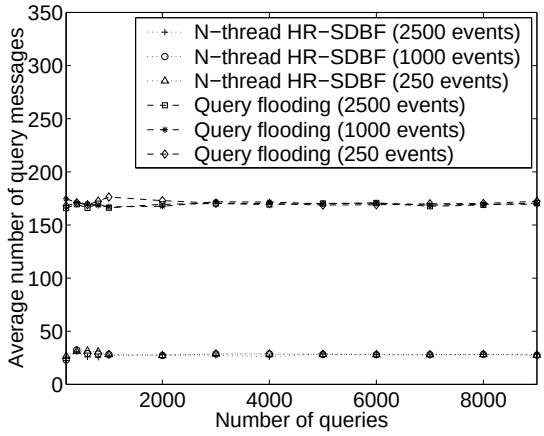
(c)

6.5 N-thread HR-SDBF vs. query flooding

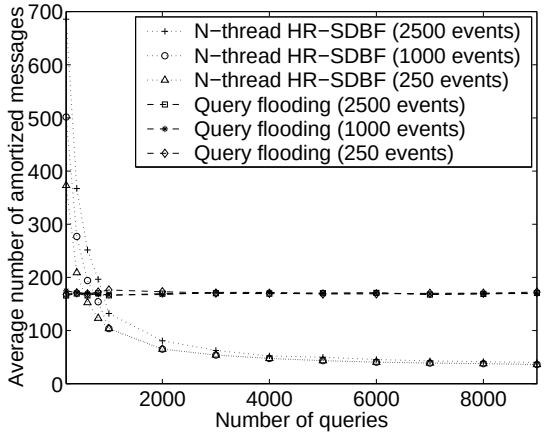
The N-thread HR-SDBF is evaluated against query flooding. Both schemes have the same query TTL, which is k, the no-loss scope in HR-SDBF. Each query is flooded within k-hops in query flooding. In N-thread HR-SDBF, the minimum hint percentage for forwarding a query is 100. A sensor forwards a query to a neighbour only if all bits of the desired event are set in that neighbour's SDBF. Queries are not forwarded outside k-hop neighbourhoods of sinks. We are interested in the number of events that both approaches find, and the query traffic and the routing traffic that both approaches generate as the number of queries changes (i.e., the event query frequency changes).

The simulation results show that both N-thread HR-SDBF and query flooding locate all events within the same k-hop neighbourhood in three event scenarios. Therefore we only plot the query traffic and the amortised traffic incurred by both schemes in Figure 18. The average number of query messages forwarded by N-thread HR-SDBF is always much smaller than query flooding, as shown in Figure 18(a) because N-thread HR-SDBF utilises hints.

Figure 18 N-thread HR-SDBF vs. query flooding, varying number of queries (R = 15) (a) average number of query messages (b) average number of amortised messages



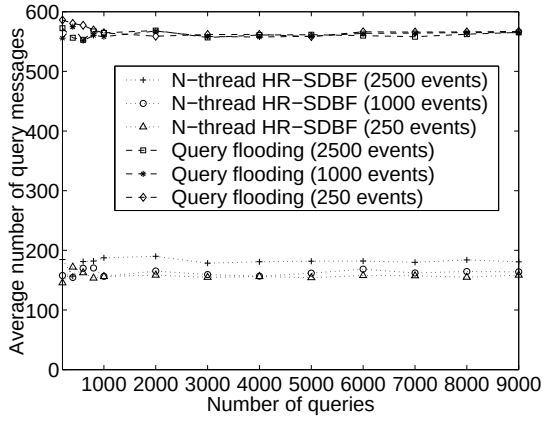
(a)



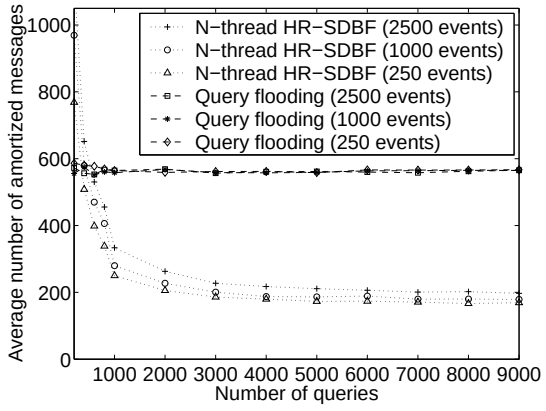
(b)

Figure 18(b) shows that query flooding has almost the same amortised traffic when the number of queries changes. *N*-thread HR-SDBF generates a larger amortised traffic than query flooding when the number of queries is small (<1000). However, the amortised traffic in *N*-thread HR-SDBF drops dramatically as the number of queries increases. When the number of queries is greater than 1,000, *N*-thread HR-SDBF incurs less amortised traffic than query flooding. The number of amortised messages delivered by *N*-thread HR-SDBF decreases slowly when the number of queries is more than 2,000. At 9,000 queries, HR-SDBF finds the same number of events with almost three times less traffic. Figure 18 also shows that the number of events does not make an impact on the performances of *N*-thread HR-SDBF and query flooding.

Figure 19 *N*-thread HR-SDBF vs. query flooding, varying number of queries ($R = 20$) (a) average number of query messages (b) average number of amortised messages



(a)



(b)

Figure 19 demonstrates the query traffic and the amortised traffic of *N*-thread HR-SDBF and query flooding in a denser network with the same configuration except that the wireless communication radius is set to 20 metres. We observe similar performance patterns. *N*-thread HR-SDBF outperforms query flooding when events are queried frequently. The average number of query messages and the average number of amortised messages of both approaches increase in the denser network because the average degree is higher. Both schemes also locate more events in the denser network because more nodes can be searched within *k*-hops.

In summary, when designing HR-SDBF, breaking ties according to the Max-Min TTL counter strategy is significantly better than random selection. The exponential decay model is slightly better than the linear decay model. One-thread HR-SDBF accomplishes a higher query success rate with the same amortised traffic than SQR. The *N*-thread HR-SDBF locates all desired events within *k*-hop neighbourhoods but incurs much less amortised traffic than query flooding when events are queried frequently.

7 Conclusions

In this paper, we proposed a routing protocol, called HR-SDBF. It routes queries based on probabilistic hints that are advertised by the proposed data structure, SDBF. An SDBF is a variant of the BF. Like a BF, an SDBF can represent a set of elements. In addition, an SDBF can describe different amounts of information about an element, denote probabilistic membership, and can be used as a channel coding scheme. The HR-SDBF protocol uses SDBFs to propagate routing hints about a set of events such that the information about an event does not attenuate within a *k*-hop neighbourhood of the event source but decreases outside the *k*-hop neighbourhood as the distance from the boundary of the *k*-hop neighbourhood increases.

In HR-SDBF, sinks can conduct two types of searches: one-thread best search or *N*-thread search. In the one-thread best search, a node always forwards a query to the best neighbour that has the most hints about a desired event. Ties can be resolved by random selection or based on the TTL counter values. In the *N*-thread search, a node directs a query to all neighbours with the full amount of information. Compared to existing query-based routing protocols in WSNs, HR-SDBF improves the query success rate with low amortised routing overhead and reduces energy consumption by keeping probabilistic hints instead of precise hints.

In the future, we plan to explore other decay models in HR-SDBF, such as decaying based on node degrees. We also intend to do analytical and simulation study in extending HR-SDBF to clustered WSNs or actor-sensor model WSNs.

Acknowledgements

This work was supported in part by NSF grants ANI 0073736, CCR 0329741, CNS 0422762, CNS 0434533 and EIA 0130806.

References

Agrawal, D.P. and Zeng, Q. (2003) *Wireless and Mobile Systems*, Thomson Brooks/Cole, Inc.

Akyildiz, I.F. and Kasimoglu, I.H. (2004) 'Wireless sensor and actor networks: research challenges', *Ad Hoc Networks Journal (Elsevier)*, October, Vol. 2, pp.351–367.

Akyildiz, I.F., Su W., Sankarasubramaniam Y. and Cayirci, E. (2002a) 'A survey on sensor networks', *IEEE Communications Magazine*, August, pp.102–114.

Akyildiz, I.F., Su, W., Sankarasubramania, Y. and Cayirci E. (2002b) 'Wireless sensor networks: a survey', *Computer Networks (Elsevier) Journal*, Vol. 38, No. 4, pp.393–422.

Al-Karaki, J.N. and Kamal, A.E. (2004) 'Routing techniques in wireless sensor networks: a survey', *IEEE Wireless Communications*, December, Vol. 11, No. 6, pp.6–28.

Bloom, B.H. (1970) 'Space/time tradeoffs in hash coding with allowable errors', *Communications of ACM*, July, Vol. 13, No. 7, pp.422–426.

Braginsky, D. and Estrin, D. (2001) 'Rumor routing algorithm for sensor networks', *Proc. of the 2001 International Conference in Distributed Computing Systems*, November.

Estrin, D., Govindan R., Heidemann J. and Kumar, S. (1999) 'Next century challenges: scalable coordination in sensor networks', *Proc. of ACM Mobi-Com'99*.

Fan, L., Cao, P., Almeida, J. and Broder, A. (1998) 'Summary cache: a scalable wide-area web cache sharing protocol', *Proc. of ACM SIGCOMM'98*, September, pp.254–265.

Ghose, A., Groklags, J. and Chuang, J. (2003) 'Resilient data-centric storage in wireless ad-hoc sensor networks', *Proc. of the 4th International Conference on Mobile Data Management (MDM 2003)*.

Guo, D., Chen, H., Luo, X. and Wu, J. (2006) 'Theory and Network Application of Dynamic Bloom Filters', *Proc. of IEEE INFOCOM'06*.

Intanagonwiwat, C., Govindan, R. and Estrin, D. (2000) 'Directed diffusion: a scalable and robust communication paradigm for sensor networks', *Proc. of ACM Mobi-Com'00*.

Kumar, A., Xu, J. and Zegura, E.W. (2005) 'Efficient and scalable query routing for unstructured peer-to-peer net-works', *Proc. of IEEE INFOCOM'05*.

Li, L., Halpern, J. and Haas, Z. (2002) 'Gossip-based adhoc routing', *Proc. of the 21st Conference of the IEEE Communications Society (INFOCOM'02)*.

Melodia, T., Pompili, D., Gungor, V.C. and Akyildiz, I.F. (2005) 'A distributed coordination framework for wireless sensor and actor networks', *Proc. of ACM Mobi-Hoc'05*.

Mitzenmacher, M. (2002) 'Compressed bloom filters', *IEEE Transactions on Networks*, Vol. 10, No. 5, pp.604–612.

Schurgers, C. and Srivastava, M.B. (2001) 'Engergy efficient routing in wireless sensor networks', *Proc. of 2001 MIL-COM Communications for Network-centric Ops: Creating the Information Force*.

Shah R.C. and Rabaey J.M. (2002) 'Energy aware routing for low energy ad hoc sensor networks', *Proc. of IEEE WCNC'02*.

Whang, K., Vander-Zanden, B. and Taylor, H. (1990) 'A linear time probabilistic counting algorithm for database applications', *ACM Transactions on Database Systems*, Vol. 15, No. 2, pp.208–229.

Ye, F., Chen, A., Lu, S. and Zhang, L. (2001) 'A scalable solution to minimum cost forwarding in large sensor networks', *Proc. of the 10th International Conference in Computer Communication Networks*.

Appendix

Proof of Proposition 1 Impact of false positives

Proof: It is assumed that in a small neighbourhood, $w_i = n_c * (n_c - 1)^{i-1}$. The simplifying assumption holds as long as $\sum_{j=1}^{i} w_i \ll N_{sn}$. Then, the total amount of hint update originated at the event source s and spread over the entire WSN is

$$\begin{aligned} U &= \sum_{i=1}^{k} w_i * d + \sum_{i=k+1}^{\infty} w_i * d * p^{i-k} \\ &\le \sum_{i=1}^{k} w_i * d + \sum_{i=k+1}^{j} w_i * d * p^{i-k} \\ &\quad + d * p^{i-j} * \left(N_{sn} - \sum_{t=1}^{j} w_t \right). \end{aligned}$$

U equals to the total hint update traffic that comes from all other sensors and flows into the sensor s because of the symmetry of the decay model. U is evenly distributed among the SDBF-EDs of the n_c neighbours of s. The number of 1s in each SDBF-ED is U/n_c. According to Whang et al. (1990) statistics, the number of 1s in an SDBF-ED, is, $m * \left(1 - e^{-U/(n_c * m)}\right)$. The percentage of bits set to 1 in an SDBF-ED, denoted by λ_p, is

$$\lambda_p = 1 - e^{-U/(n_c * m)}$$

The number of bits set to 1 because of false positives in an SDBF-ED is a random variable, F, which follows a binomial distribution $B(d, \lambda p)$. □

Proof of Proposition 2 State transition probability matrix C

Proof: If a sensor is within the no-decay k-hop neighbourhood of an event source, it always forwards a query for that event to its upstream neighbour on its shortest path to the event source because the hint about the event is precise within the k-hop neighbourhood. The transition probability from state k to itself is 1 and to any other state 0. Because a sensor cannot forward a query to another sensor on its shortest path that is 2 or more hops closer to the event source, the transition probability from state i to state $j \le i - 2$ is 0. To compute a_i, we need to know the amount of information for an event coming from the upstream neighbour on the shortest path at state i. In the SDBF-ED, the hint, initially d bits, decays exponentially at rate $p \in (0, 1]$ outside the k-hop neighbourhood. The probability of a bit at distance i from the event source remaining 1 is p^{i-k}. In the SDBF-ED from the upstream neighbour on the shortest path, each of the d bits survives with a probability, $\lambda_s = 1 - (1 - p^{i-k})(1 - \lambda_p) = p^{i-k} + \lambda_p - \lambda_p * p^{i-k}$. Therefore, the number of bits set to 1 among the d bits of the SDBF-ED from the upstream neighbour is a binomial random variable, Z, which follows the distribution $B(k, \lambda_s)$.

a_i is the sum of the probabilities in the following two cases.

1 The hint from the upstream neighbour is greater than that from any downstream neighbour. The query is definitely forwarded to the upstream neighbour. The probability that this case occurs is $p_{i,1} = P\left[\left(Z > F\right)^{n_c - 1}\right]$.

2 The hint from the upstream neighbour is the same as u downstream neighbours but greater than all others. The query is forwarded to the upstream neighbour with probability $1/(u + 1)$. This case happens with a probability $p_{i,2} = \sum_{u=1}^{n_c - 1} P\left[\left(Z = F\right)^u \left(Z > F\right)^{n_c - u - 1}\right]$.

Therefore, $a_i = p_{i,1} + p_{i,2}$.

A sensor transits from state i to state $j > i$ if the number of false positives caused by a non-upstream neighbour is greater than the hints from the upstream neighbour. All neighbours generate uniformly random false positives. The probability of state i to all states $j > i$, denoted by b_i, is $b_i = \sum_{j=1}^{k+5} b_{ij} = 1 - a_i . b_{ij} = \left(ws_i \Big/ \sum_{t=k+1}^{k+5} ws_t\right)\left(1 - a_i\right)$. □