Continuous Probabilistic Count Queries in Wireless Sensor Networks

Anna Follmann*, Mario A. Nascimento†, Andreas Zuefle*,
Matthias Renz*, Peer Kröger* and Hans-Peter Kriegel*
*Department of Computer Science, Ludwig-Maximilians-Universität, München, Germany
†Department of Computing Science, University of Alberta, Edmonton, Canada
follmann@cip.ifi.lmu.de, mn@cs.ualberta.ca, {zuefle, kroegerp, renz, kriegel}@dbs.ifi.lmu.de

Abstract—Count queries in wireless sensor networks report the number of sensor nodes for which the measured values satisfy a given query predicate. However, measurements in wireless sensor networks are typically imprecise due to limited accuracy of the sensor hardware or fluctuations in the observed environment. Consequently, queries performed on these imprecise information implicate imprecise answers. In this paper, we study the problem of computing continuous probabilistic count queries in a distributed system, i.e., given a query Q we compute a probability distribution over the number of sensors satisfying Q's predicate. Such queries enables us to compute the probability that *exactly*, *at most* or *at least* k nodes satisfy Q. We investigate four algorithms that efficiently compute probabilistic count queries in a centralized manner as well as in-network and/or incrementally. In our performance evaluation we investigate all proposed algorithms in terms of the number of sent messages and show that our incremental approach is able to produce up to 80% less message transfers compared to the centralized algorithm.

1 Introduction

A wireless sensor network (WSN) is usually defined as a set of spatially distributed autonomous sensors that cooperatively monitor physical or environmental conditions in an area of interest, e.g., [1], [2]. A single sensor node consists of one (or more) sensors, a microprocessor, a small amount of memory, a radio transceiver and a battery [3]. However, measurements by nodes in WSNs are typically imprecise, be it because of the sensor's hardware or because of fluctuations in the environment itself [4]. Processing uncertain data sets leads to a variety of novel problems and demands more complex query algorithms. Typically, queries on uncertain data known as probabilistic queries involve computation of probability distributions over possible answers. In addition, if we want to process probabilistic queries in wireless sensor networks we have to consider the general characteristics and limitations of such networks. Even if sensors are gaining in computing ability they remain constrained by limited batteries. With communication being the primary drain on power it is crucial that we reduce transmissions to extend the lifetime of the network [3].

Technical Report TR 10-09. December 2010. Dept. of Computing Science. University of Alberta. Canada. All rights reserved. Submitted for publication.

TABLE 1
Example of probabilistic sensor readings

SID	Location	Timestamp	Temperature	Probability
s_1	Room 101	11:40	50	0.2
s_2	Room 102	11:40	35	0.8
s_3	Room 103	11:40	38	0.7
s_4	Room 203	11:40	40	0.4
s_1	Room 101	11:50	–	0.0
s_2	Room 102	11:50	35	0.8
s_3	Room 103	11:50	38	1.0
s_4	Room 203	11:50	40	0.4

This paper addresses count queries in WSNs. Count queries in sensor networks are very useful for many applications, for example, if we want to control the climate of a building complex using a number of sensors monitoring the current temperature at different locations within the building. Count queries can be used to adjust the settings of the air conditioning system or automatic shutters for specific conditions. We could for example turn on the heating if *exactly* k, *at most* k or *least* k sensors measure a temperature below (or above) a specific threshold temperature. Traditional count queries simply count the number of sensors that satisfy the given query predicate and return the value of the counter. In fact, in-network aggregation can be applied easily to optimize the query's energy cost [5]. However, adding uncertainty raises new issues for the processing of the query itself as well as for the aggregation strategies that can be applied. Instead of simply counting sensors fulfilling the query predicate, now, for each sensor we have to consider the probability that it satisfies the query predicate. Thus, instead of returning the number of sensors that satisfy the query, we now have a probability value that a specific number of sensors satisfies the query. It can intuitively be understood that the result of such a probabilistic count query is in fact a probability distribution.

Consider the example in Table 1. Let $S = \{s_1, s_2, s_3, s_4\}$

be a WSN with four sensors monitoring a building. Sensors s_1, s_2 and s_3 are installed on the first floor and s_4 is placed on the second floor. They measure the temperature and send their data periodically (for example every 10 min.). Every sensor reading contains a location, a temperature value, a specific time point as well as a probability of satisfying a query, e.g. *"Temperature equal to exactly* $35°$*"*. Table 1 shows example readings of the four sensors in such application. Examples for probabilistic count queries on this table could be "What is the probability that exactly 2 sensors satisfy the query?" or "What is the probability that *at least* (*at most*) 2 sensors satisfy the query?" As our main contribution in this paper we investigate four algorithms to answer these types of queries within WSNs with the ultimate goal of minimizing the energy cost of processing such queries.

The remainder of this paper is organized as follows. In Section Section 2 we briefly discuss related work. A formal description of our data model and query computation is given in Section 3. In Section 4 we present four algorithms for solving the problem in a centralized manner as well as in-network and/or incrementally. We experimentally evaluate the efficiency of the proposed algorithms in Section 5 and conclude the paper in Section 6. The notations that we use in this paper are summarized in Table 3.

2 Related Work

2.1 Probabilistic Data and Query Answering

Due to the steady increase in the number of application domains where uncertain data arises naturally, such as data integration, information extraction, sensor networks, persuasive computing etc., modelling and querying probabilistic, uncertain, incomplete, and/or fuzzy data in database systems is a fast growing research direction [6], [7], [8], [9]. Previous work has spanned a range of issues from theoretical development of data models and data languages [9], to practical implementation issues such as indexing techniques, e.g. [7], [10] and probabilistic similarity query techniques [11]. Uncertainty is either modelled as tuple level uncertainty, where "existence" probabilities are attached to each tuple, or as attribute-level uncertainty, where (discrete or continuous) probability distributions are attached to the attributes. In this paper we adopt the uncertainty model proposed in [8]. The proposed approaches differ further based on whether they consider correlations or not. Most work in probabilistic databases has either assumed independence [9] or has restricted correlations that can be modelled [8] and more recently, arbitrary correlations [12].

Cheng et *al.* [6] provided a general classification of probabilistic queries and evaluation algorithms over uncertain data sets. In [13] Ross et. *al.* addressed the problem of answering probabilistic count queries. However, they proposed a solution that requires the individual consideration of each possible world. This implies a computational cost that is exponential in the number of sensor nodes.

The problem of answering probabilistic count queries is related to the problem of answering probabilistic top-k queries, which was recently studied with great interest [11], [12], [14]–[16]. In order to determine the rank of an uncertain tuple t, the number of tuples which have a score higher than t needs to be counted. In this work, we will generalize efficient techniques used to solve the probabilistic ranking of uncertain objects in the context of databases in order to apply these techniques to answer probabilistic count queries on sensor networks where usually completely different parameters need to be optimized.

2.2 Wireless Sensor Networks

Wireless sensor networks are studied in their various aspects with work ranging from optimization [17], over practical implementation issues [18], to experimental analyzation [19]. In [2] Akyildiz et. *al.* summarize the characteristics of WSN. However previous work considers data to be certain. To the best of our knowledge Wang et *al.* are the only ones who studied the field of probabilistic queries in a distributed system such as a wireless sensor network. In [20], they address the problem of answering probabilistic top-k queries in wireless sensor networks. In this paper we address the related problem of answering probabilistic count queries in a wireless sensor network. In addition to previous approaches, we introduce update strategies to handle the continuous stream of data that is produced by the sensor network.

3 Background

3.1 Probabilistic Data Model

Given a WSN ($S = \{s_1, s_2, \ldots, s_n\}$) composing a set of n sensors yielding n sensor values[1] at a given time t. For a given query Q, each sensor s_i has a probability $P^Q_{s_{i,t}}$ of satisfying Q's predicate at time t. Consider the temperature monitoring application of our first example. Each tuple in Table 1 consists of a sensor id, a location reading, a time stamp t, a temperature value and a probability value $P^Q_{s_{i,t}}$ indicating the likelihood that sensor s_i satisfies the predicate of a given query Q at time point t[2]. Thereby we assume data vectors of two sensors $s_x \neq s_y$ to be mutually independent, i.e. each of the probability values assigned to the sensors is an independent Bernoulli random variable with $P(X_i = 1) = 1 - P(X_i = 0) = P^Q_{s_{i,t}}$.

For solving probabilistic queries on our uncertain sensor network model, we apply the possible worlds semantics model which was originally proposed by Kripke [21] for modal logics and is commonly used for

1. In the remainder we use the term sensor to refer to the corresponding sensor value

2. The predicate of Q is irrelevant for our observations. In the following we assume a given Q as a base query on top of which we can later build our actual probabilistic count queries.

TABLE 2
Possible Worlds of Table 1

Possible World $W_{k,j}$	Probability $P(W_{k,j})$
$W_{4,1} = \{s_1, s_2, s_3, s_4\}$	0.2*0.8*0.7*0.4 = 0.0448
$W_{3,1} = \{s_1, s_2, s_3\}$	0.2*0.8*0.7*(1-0.4) = 0.0672
$W_{3,2} = \{s_1, s_2, s_4\}$	0.2*0.8*(1-0.7)*0.4 = 0.0192
$W_{3,3} = \{s_1, s_3, s_4\}$	0.2*(1-0.8)*0.7*0.4 = 0.0112
$W_{3,4} = \{s_2, s_3, s_4\}$	(1-0.2)*0.8*0.7*0.4 = 0.1792
$W_{2,1} = \{s_1, s_2\}$	0.2*0.8*(1-0.7)*(1-0.4) = 0.0288
$W_{2,2} = \{s_1, s_3, \}$	0.2*(1-0.8)*0.7*(1-0.4) = 0.0168
$W_{2,3} = \{s_1, s_4\}$	0.2*(1-0.8)*(1-0.7)*0.4 = 0.0048
$W_{2,4} = \{s_2, s_3\}$	(1-0.2)*0.8*0.7*(1-0.4) = 0.2688
$W_{2,5} = \{s_2, s_4\}$	(1-0.2)*0.8*(1-0.7)*0.4 = 0.0768
$W_{2,6} = \{s_3, s_4\}$	(1-0.2)*(1-0.8)*0.7*0.4 = 0.0448
$W_{1,1} = \{s_1\}$	0.2*(1-0.8)*(1-0.7)*(1-0.4) = 0.0072
$W_{1,2} = \{s_2\}$	(1-0.2)*0.8*(1-0.7)*(1-0.4) = 0.1152
$W_{1,3} = \{s_3\}$	(1-0.2)*(1-0.8)*0.7*(1-0.4) = 0.0672
$W_{1,4} = \{s_4\}$	(1-0.2)*(1-0.8)*(1-0.7)*0.4 = 0.0192
$W_{0,1} = \emptyset$	(1-0.2)*(1-0.8)*(1-0.7)*(1-0.4) = 0.0288

representing knowledge with uncertainties. However, there have been different adaptations of the model for probabilistic databases [22], [9], [8]. Here, we use the model as proposed in [8], specifically, a possible world is a set of sensors satisfying Q associated with the probability that this world is true at a certain time t. In particular, we define $W_{k,j}$ as the j^{th} world where exactly k sensors satisfy Q at time t. The probability $P(W_{k,j})$ of a possible world $W_{k,j}$ at time t is computed by multiplying $P^Q_{s_i,t}$ for each sensor $s_i \in W_{k,j}$. For our four entries in the Table 1 there are $2^4 = 16$ possible worlds in total. Table 2 displays those possible worlds and their respective probabilities.

3.2 Probabilistic Count Query

Given the WSN with the set of uncertain sensors S and any query Q as described above and a count parameter k. The problem to be solved for a probabilistic count query is to compute the probability $P^t(k, S, Q)$ that exactly k sensors in the network S satisfy Q at time point t. We call $P(k, S, Q)$ *probabilistic count*. Instead of a single probabilistic count, a probabilistic count query returns the probability distribution of $P^t(k, S, Q)$ over k ($0 \leq k \leq |S|$) called *count histogram*. To lighten the notation, we use P_{s_j} equivalent for $P^Q_{s_j,t}$ and $P(k, S, Q)$ equivalent for $P^t(k, S, Q)$, i.e., all probabilities and data are considered with respect to the same point in time.

The distribution of the count probabilities depends on the probability values P_{s_i}. However, as we shall see in the discussion that follows the computation of the distribution does not depend on the actual probability values.

A naive method to answer a probabilistic count query is to first enumerate all possible worlds of S and then sum up the probabilities of those worlds where exactly k tuples appear, that is:

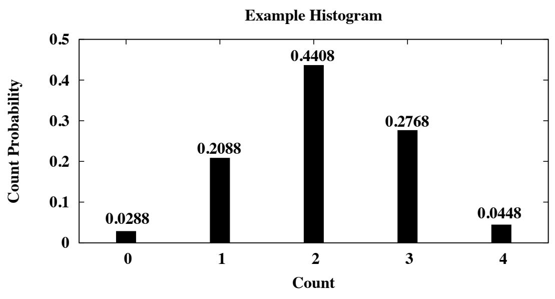


Fig. 1. Probability Count Histogram for Example 1

TABLE 3
Description of the notations

Symbol	Description of Symbol
Q	Query
k	Count
t	Time Point
S	Set of Sensors
s_j	Sensor
S_j	Subset of sensors $S_j = S \setminus s_j$
$P^Q_{s_j,t}$	Probability of s_j
$P^t(k, S, Q)$	Probability that k ($0 \leq k \leq \|S\|$) sensors in S satisfy Q
$P^t(k, S_j, Q)$	Probability that k ($0 \leq k \leq \|S\|$) sensors in a subset $S_j = S \setminus s_j$ fulfill Q

$$P(k, S, Q) = \sum_j P(W_{k,j}) \tag{1}$$

Considering Table 1 and a query Q at t = 11:40, we can intuitively compute the probabilities of all possible worlds and then sum up the possible worlds that contain the same amount of sensors by using Equation 1. Figure 1 shows the resulting count histogram. Since the number of possible worlds is exponential in the number of sensors this naive method is not very efficient.

In the following, we show how to efficiently compute a probabilistic count query in general, before we address the problem of answering *continuous* probabilistic count queries in *wireless sensor networks* in Section 4. In [4], Hua et *al.* have proposed an efficient algorithm based on the *Poisson Binomial Recurrence* to avoid searching all possible worlds in a different setting, however, the problem description and definition given there can be adjusted to fit our problem.

3.3 Poisson Binomial Recurrence

Let S be a set of sensors and the order of the sensors $s_j \in S$ is irrelevant. Every sensor s_j satisfies Q with a probability P_{s_j}. The probabilistic count $P(k, S, Q)$ is the probability that k ($0 \leq k \leq |S|$) sensors in S satisfy Q's predicate. Moreover, the subset probability $P(k, S_j, Q)$ is the probability that k sensors in the subset $S_j = S \setminus s_j$ fulfill Q. Then, the probabilistic count $P(k, S, Q)$ depends only on the $(k-1)$ other tuples in the subset S_j. That is, k sensors in S satisfy Q only when s_j satisfies Q and at the same time $k-1$ sensors in S_j satisfy Q or when s_j does not satisfy Q and at the same time k sensors in S_j satisfy Q. Defining base cases $P(0, \emptyset, Q) = 1$, $P(j, \emptyset, Q) = 0$ for $(0 \leq j \leq |S|)$ and $P(-1, S, Q) = 0$ for any set of sensors. Then,

$$\begin{aligned} P(j, S, Q) = P(j-1, S_j, Q)P_{s_j} \\ + P(j, S_j, Q)(1 - P_{s_j}) \end{aligned} \tag{2}$$

Equation 2 can be iteratively applied to compute the result of a probabilistic count query. In fact, in [23] (Sec. 1.7) this is called the *Poisson Binomial Recurrence*, and it is shown that:

$$P(0, S, Q) = P(0, S_j, Q)(1 - P_{s_j}) = \prod_{j=1}^{k}(1 - P_{s_j}), \text{and}$$
$$P(j, S, Q) = P(j-1, S_j, Q)P_{s_j} + P(j, S_j, Q)(1 - P_{s_j})$$

for $0 < j \leq |S|$.

In each iteration we can omit the computation of any $P(j, S, Q)$ where $j \geq k$, since we are not interested in any counts other than k and do not need them to process the query. In total, for each $0 \leq j < k$ and each of the n sensors $s_i \in S$, $P(j, S, Q)$ has to be computed resulting in $O(k \cdot n)$ time complexity.

Example 1. *Consider the sensor readings of our example application in Section 1 with a query Q at time $t = 11:40$ and $k = 2$. Assuming that all tuples satisfy Q's predicate with the probabilities listed in Table 1, we have $P_{s_1} = 0.2$, $P_{s_2} = 0.8$, $P_{s_3} = 0.7$ and $P_{s_4} = 0.4$. Note ,that the order in which we process the sensor readings is irrelevant. For the sake of simplicity, we process the table line by line starting at the top. Using Equation 2 we have $S = \{s_1\}$, $S_1 = \emptyset$ and*

$$P(0, S, Q) = P(-1, S_1, Q)P_{s_1} + P(0, S_1, Q)(1 - P_{s_1}) = 0.8$$
$$P(1, S, Q) = P(0, S_1, Q)P_{s_1} + P(1, S_1, Q)(1 - P_{s_1}) = 0.2$$

Next we process s_2 by adding another iteration of Equation (2). With $S = \{s_1, s_2\}$ and $S_2 = \{s_1\}$ we obtain the following:

$$\begin{aligned} P(0, S, Q) &= P(-1, S_2, Q)P_{s_2} + P(0, S_2, Q)(1 - P_{s_2}) \\ &= 0.16, \\ P(1, S, Q) &= P(0, S_2, Q)P_{s_2} + P(1, S_2, Q)(1 - P_{s_2}) \\ &= 0.68, \\ P(2, S, Q) &= P(1, S_2, Q)P_{s_2} + P(2, S_2, Q)(1 - P_{s_2}) \\ &= 0.16. \end{aligned}$$

With s_3 and $S = \{s_1, s_2, s_3\}$ we have:

$P(0, S, Q) = 0.048$, $P(1, S, Q) = 0.316$, $P(2, S, Q) = 0.524$.

Since $k = 2$ we can stop at that point and do not need to compute $P(3, S, Q)$. Finally, we add s_4 ($S = \{s_1, s_2, s_3, s_4\}$):

$$P(0, S, Q) = 0.0288,\ P(1, S, Q) = 0.2088,$$
$$P(2, S, Q) = 0.4408$$

and can return the result of our query $P(2, S, Q) = 0.4408$.

As illustrated in Example 1, Equation 2 can be used to efficiently compute the probabilistic count $P(k, S, Q)$ that k sensors satisfy a query Q. Hence, we can easily compute the probability, that *at most* or *at least* k sensors satisfy Q's predicate. To compute the probability that *at most* k sensors satisfy a query Q, we intuitively sum up all probabilistic counts $P(j, S, Q)$ with $0 \leq j \leq k$.

$$P_{-}(k, S, Q) = \sum_{j=0}^{k} P(j, S, Q) \tag{3}$$

To compute the probability that *at least* k sensors satisfy Q it is useful to know that the values of a complete count histogram sum up to 1. Hence, we can compute the probability of *at least* k sensors satisfying Q by using Equation 3 as follows:

$$P_{+}(k, S, Q) = 1 - \sum_{j=0}^{k-1} P(j, S, Q) \tag{4}$$

3.4 Allowing Certainty

Until now, we assumed all sensors to be uncertain at all times ($0 < P_{s_j} < 1$). But certain circumstances or queries call for the existence of certain values. On one hand, sensors can die or wake up from one round to another, messages can get lost or there can be restrictions to the query, i.e. we only want to query sensors in a bounded area. Thus, particular sensors do not participate in a query at all, and we can safely set their probability P_{s_j} to *zero*. On the other hand there could be reasons to set the probability P_{s_j} to *one*, e.g., if a sensor's samplings are very stable over a sufficient period of time. In the following, we explore the effects of $P_{s_j} = 0$ and $P_{s_j} = 1$.

3.4.1 Effect of $P_{s_j} = 0$

Reconsider Example 1. We now add a fifth sensor s_5 with $P_{s_5} = 0$ and observe the effect of a *zero probability*. After processing the four sensors in Example 1 we had $P(0, S_4, Q) = 0.048$, $P(1, S_4, Q) = 0.316$ and $P(2, S_4, Q) = 0.524$. We now incorporate P_{s_5} and compute the probability that exactly two sensors satisfy the query predicate (c.f. Equation 2).

$$\begin{aligned} P(2, S, Q) &= P(1, S_4, Q)P_{s_4} + P(2, S_4, Q)(1 - P_{s_4}) \\ &= 0.316 \cdot 0 + 0.524 \cdot 1 = 0.524 \\ &= P(2, S_4, Q). \end{aligned}$$

Apparently, the incorporation of a value $P_{s_j} = 0$ does not affect $P(j, S, Q)$ $(0 \leq j \leq k)$ at all. The following lemma formalizes this observation.

Lemma 1. *Let $0 \leq k \leq n, P_{s_j} = 0$. It holds that*

$$\forall k : P(k, S, Q) = P(k, S \setminus s_j, Q).$$

Proof: Using Equation 2 we obtain:

$$\begin{aligned} P(k, S, Q) &= P(k-1, S \setminus s_j, Q) \cdot 0 \\ &+ P(k, S \setminus s_j, Q)(1 - 0) \\ &= P(k, S \setminus s_j, Q). \end{aligned}$$

□

Thus, Lemma 1 allows us to ignore any sensor s_j with $P_{s_j} = 0$ in the computation of the Poisson binomial recurrence.

3.4.2 Effect of $P_{s_j} = 1$

Again, we use the example of the previous section, but now assuming $P_{s_5} = 1$. The probability that exactly two sensors satisfy the query predicate is derived by using Equation 2:

$$\begin{aligned} P(2, S, Q) &= P(1, S_4, Q)P_{s_4} + P(2, S_4, Q)(1 - P_{s_4}) \\ &= 0.316 \cdot 1 + 0.524 \cdot 0 = 0.316 \\ &= P(1, S_4, Q). \end{aligned}$$

The incorporation of of a value $P_{s_j} = 1$ shifts all values in the Poisson Binomial Recurrence to the right, formalized by the following lemma:

Lemma 2. *Let* $0 \leq k \leq n, P_{s_j} = 1$. *It holds that*

$$\forall k : P(k, S, Q) = P(k - 1, S \setminus s_j, Q).$$

Proof: Using Equation 2 we obtain:

$$\begin{aligned} P(k, S, Q) &= P(k - 1, S \setminus s_j, Q) \cdot 1 \\ &+ P(k, S \setminus s_j, Q)(1 - 1) \\ &= P(k - 1, S \setminus s_j, Q). \end{aligned}$$

□

Lemma 2 allows us to avoid iterations of the Poisson binomial recurrence for each sensor s_j with $P_{s_j} = 1$. Instead we can use a counter which is incremented by one for each such s_j, thus counting the number of positions that the Poisson Binomial Recurrence has to be shifted to the right.

In summary, Lemma 1 and Lemma 2 allow us to handle zero and one values in a very efficient way.

4 Probabilistic Count Queries in Wireless Sensor Networks

In the previous section there was no concern as to the fact that probabilities are to be calculated within a wireless sensor network. As mentioned in Section 1 it is crucial for applications on WSNs to reduce energy cost, and this mainly achieved through reducing communication. Therefore, we must consider the typical underlying characteristics of WSNs such as network topology, routing and scheduling. In the following we propose four algorithms which solve the problem of answering continuous count queries in a WSN. Thus, we now take the local distribution of data as well as the temporal dimension into consideration and reinstate the notation $P^t(k, S, Q)$. We assume that the nodes in S are connected together via a logical tree where the sink node (or base-station) is the tree's root. The choice of the tree's topology does matter, but is outside the scope of this paper. For the sake of simplicity, we assume it to be a hop-based shortest path tree commonly used in other works, e.g., [5].

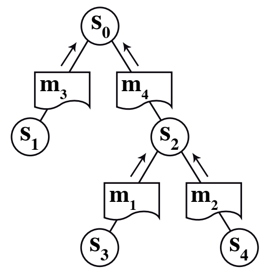


Fig. 2. Example Network Structure

4.1 A Centralized Algorithm

In the centralized approach all probability values are sent to the root node at every round without previous processing. Thus, the sink node can be seen as a central database that receives and temporarily stores the readings of all sensor nodes within the network and that centrally processes queries on the WSN. The probabilistic count histogram can then be easily computed by using Equation 2.

While leaf nodes send only one value to their parent node, intermediate nodes send their own value plus all values received from their child nodes. Thus, the payload size of the packages that are sent within the network increases as we get closer to the root node. With $0 < P_{s_j} < 1$ for all sensors and unlimited payload size for any message sent within the network, $n-1$ messages are sent to the root in every round. However, in reality, the number of messages is likely to be much larger, depending on the topology and the fixed payload size.

With the knowledge that we gained in Section 3.4, aggregation strategies can be applied whenever a sensor has a *zero probability* or a *one probability*. According to Lemma 1, all sensors s_j with $P_{s_j} = 0$ can safely be ignored in the computation. For sensors s_j with $P_{s_j} = 1$, Lemma 2 applies and we use a counter variable C_1^t that denotes the number of such tuples. Thus, Equation 2 is only required for sensors s_j for which $0 < P_{s_j} < 1$, allowing us to have shorter messages, thus saving energy in packet transmission.

Example 2. *Consider Example 1 and Figure 2. Sensors s_i are depicted as circles, m_i denotes a message sent from one node to the other with i indicating the ordering among messages. In the first round (t = 11:40) all sensors report uncertain values. At time t = 11:50, s_1 has* zero probability *and therefore no message is sent at all. Whereas sensor s_3 is certain to satisfy Q with probability $P_{s_3} = 1$. Instead of sending a value, s_3 increments its counter ($C_1^t = 1$) and sends the counter only. Sensor s_4 sends its value as usual and sensor s_2 forwards the two probability values as well as the counter of to the sink. With two iterations of Equation 2 we obtain for t =11:50:*

$P^t(0, S, Q) = 0.12$, $P^t(1, S, Q) = 0.56$, $P^t(2, S, Q) = 0.32$ *and* $C_1^t = 1$.

4.2 A Centralized Incremental Algorithm

The naive solution works well if the probability values P_{s_j} change often from one round t to the next round $t+1$. However, this is expensive in terms of messages sent when only a few values change. In some cases it is more efficient to send all probability values and process the query as proposed in Section 4.1 only in the initial computation. In all subsequent rounds, only sensors that have changed their probability will send an update.The sink node will then compute the new count probability incrementally as described next.

We can update $P^t(k, S, Q)$ using the results of previous iterations. Let s_x be a sensor and let $P^Q_{s_{x,t}}$ and $P^Q_{s_{x,t+1}}$ denote the previous and new probability that s_x satisfies Q, respectively. Our update algorithm has two phases, summarized next:

- **Phase 1:** First we remove the effect that $P^Q_{s_{x,t}}$ had on the previous probabilistic counts $P^t(j, S, Q)$, $0 \leq j \leq k$. This yields an intermediate result of the probabilistic counts $\hat{P}^t(j, S, Q)$, $0 \leq j \leq k$.
- **Phase 2:** Next we incorporate the new probability $P^Q_{s_{x,t+1}}$ by adding it to the temporary probabilistic counts $\hat{P}^t(j, S, Q)$, $0 \leq j \leq k$ using Equation (2).

Phase 1. The following cases have to be considered:

Case 1: $P^Q_{s_{x,t}} = 0$. In this case, nothing has to be done to remove the effect of $P^Q_{s_{x,t}} = 0$ and $\hat{P}^t(j, S, Q) = P^t(j, S, Q)$.

Case 2: $P^Q_{s_{x,t}} = 1$. When $P^Q_{s_{x,t}} = 1$ we must decrement the counter C^t_1 by one. Thus, $\hat{P}^t(j, S, Q) = P^t(j, S, Q)$ and $\hat{C}^t_1 = C^t_1 - 1$.

Case 3: $0 < P^Q_{s_{x,t}} < 1$. To remove the effect of any probability $P^Q_{s_{x,t}}$ from all $P^t(j, S, Q)$, $(0 \leq j \leq k)$ we look at its incorporation via Equation 2:

$$P^t(j, S, Q) = \hat{P}^t(j-1, S_x, Q)P^Q_{s_{x,t}} + \hat{P}^t(j, S_x, Q)(1-P^Q_{s_{x,t}})$$

We can remove the effect of $P^Q_{s_{x,t}}$ by resolving Equation 2 as follows:

$$\hat{P}^t(j, S, Q) = \frac{P^t(j, S, Q) - \hat{P}^t(j-1, S, Q) \cdot P^Q_{s_{x,t}}}{1 - P^Q_{s_{x,t}}} \quad (5)$$

Since any $P^t(-1, S, Q) = 0$ we have:

$$\hat{P}^t(0, S, Q) = \frac{P^t(0, S, Q)}{1 - P^Q_{s_{x,t}}} \quad (6)$$

for $j = 0$ and can step by step compute $\hat{P}^t(j, S, Q)$ by using $\hat{P}^t(j-1, S, Q)$ and Equation (5) for any $0 < j \leq k$.

Phase 2. In Phase 2 we have to consider the same cases as in Phase 1:

Case 1: $P^Q_{s_{x,t+1}} = 0$ has no influece on the result at time $t+1$ and $P^{t+1}(j, S, Q) = \hat{P}^t(j, S, Q)$.

Case 2: $P^Q_{s_{x,t+1}} = 1$. When $P^Q_{s_{x,t}} = 1$ we must increment the counter C^t_1 by one. Thus, $\hat{P}^t(j, S, Q) = P^t(j, S, Q)$ and $\hat{C}^t_1 = C^t_1 + 1$.

Case 3: $0 < P^Q_{s_{x,t+1}} < 1$. We can incorporate the new probability $P^Q_{s_{x,t+1}}$ by an additional iteration of Equation (2).

$$P^{t+1}(j, S, Q) = \hat{P}^t(j-1, S, Q)P^Q_{s_{x,t+1}} + \hat{P}^t(j, S, Q)(1-P^Q_{s_{x,t+1}})$$

This means that, whenever a sensor sends an update, it has to send both its previous probability and its new probability. Thus, in the worst case we send twice the number of values compared to the centralized algorithm. On the other hand, the less updates are sent, the better the results for the incremental approach.

Regarding the computational complexity, the following holds for both, Phase 1 and Phase 2: Case 1 and 2 have a cost of $O(1)$ since either nothing has to be done, or C^t_1 has to be incremented or decremented. Case 3 has a total cost of $O(k)$ leading to a total execution time of $O(k)$ per old-new value pair in the root node.

Example 3. *Reconsider Example 1 where at time t = 11:40 $P^Q_{s_{1,t}} = 0.2$, $P^Q_{s_{2,t}} = 0.8$, $P^Q_{s_{3,t}} = 0.7$ and$P^Q_{s_{4,t}} = 0.4$. At time t+1 =11:50 only s_1 and s_3 change their values: $P^Q_{s_{1,t+1}} = 0.0$ and $P^Q_{s_{3,t+1}} = 1.0$.*

We start with $P^t(0, S, Q) = 0.0288$, $P^t(1, S, Q) = 0.2088$, $P^t(2, S, Q) = 0.4408$ and $C^t_1 = 0$. First, we remove the effect of $P^Q_{s_{1,t}}$ by using Equations 5 and 6:

$$\hat{P}^t(0, S, Q) = \frac{P^t(0, S, Q)}{1 - P^Q_{s_{1,t}}} = 0.036$$

$$\hat{P}^t(1, S, Q) = \frac{P^t(1, S, Q) - \hat{P}^t(0, S, Q) \cdot P^Q_{s_{1,t}}}{1 - P^Q_{s_{1,t}}} = 0.252$$

$$\hat{P}^t(2, S, Q) = \frac{P^t(2, S, Q) - \hat{P}^t(0, S, Q) \cdot P^Q_{s_{1,t}}}{1 - P^Q_{s_{1,t}}} = 0.488$$

Next we incorporate the new probability of s_1 but notice that $P^Q_{s_{1,t+1}} = 0$, so Phase 2 can be skipped. We go on with removing the effect of $P^Q_{s_{3,t}}$ and obtain:

$$\hat{P}^t(0, S, Q) = \frac{P^t(0, S, Q)}{1 - P^Q_{s_{1,t}}} = 0.12$$

$$\hat{P}^t(1, S, Q) = \frac{P^t(1, S, Q) - \hat{P}^t(0, S, Q) \cdot P^Q_{s_{1,t}}}{1 - P^Q_{s_{1,t}}} = 0.56$$

$$\hat{P}^t(2, S, Q) = \frac{P^t(2, S, Q) - \hat{P}^t(0, S, Q) \cdot P^Q_{s_{1,t}}}{1 - P^Q_{s_{1,t}}} = 0.32$$

Since the new probability of s_3 is $P^Q_{s_{3,t+1}} = 1$ we only need to increment the counter $C^t_1 = 1$.

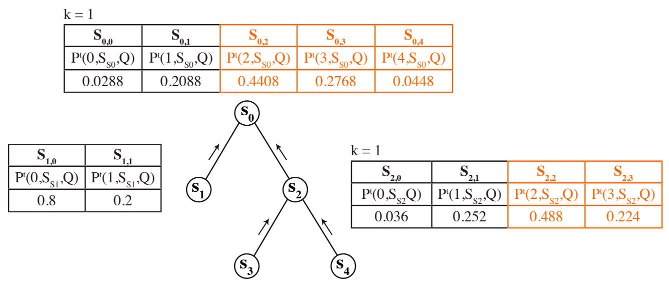

k = 1
$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$ $S_{0,4}$
$P^t(0,S_{S0},Q)$ $P^t(1,S_{S0},Q)$ $P^t(2,S_{S0},Q)$ $P^t(3,S_{S0},Q)$ $P^t(4,S_{S0},Q)$
0.0288 0.2088 0.4408 0.2768 0.0448
$S_{1,0}$ $S_{1,1}$
$P^t(0,S_{S1},Q)$ $P^t(1,S_{S1},Q)$
0.8 0.2
k = 1
$S_{2,0}$ $S_{2,1}$ $S_{2,2}$ $S_{2,3}$
$P^t(0,S_{S2},Q)$ $P^t(1,S_{S2},Q)$ $P^t(2,S_{S2},Q)$ $P^t(3,S_{S2},Q)$
0.036 0.252 0.488 0.224
s_0
s_1
s_2
s_3
s_4

Fig. 3. Multiplication of two Count Histograms

4.3 An In-Network Algorithm

In both previous algorithms the sink computes the histogram in a centralized manner. Despite the counter, there is no aggregation strategy applied. But we can profit by computing the histogram at intermediate nodes as we send the values up to the root node. On the one hand we can decrease the number of messages sent. On the other hand, intermediate count histograms could be used to query subtrees or apply early stopping conditions if a subtree satisfies the query [4].

Like in the centralized algorithm every sensor sends its value in every round. The idea is that every intermediate node s_j computes the probability histogram of its subtree by pairwise multiplying the probability histograms of its child nodes and its own probability $P_{s_j}^Q$ on the fly. *Zero probabilities* and *one probabilities* are processed as usual. As we only need the first k count probabilities to answer a query, it is sufficient to compute only the first k count probabilities and then forward them. Hence, a maximum number of $k+1$ values per message is sent at every hop up to the sink and we can gain compared to the centralized approach as the maximum payload size gets smaller.

Each probability P_{s_i} can be modelled as a minimal probabilistic count histogram with $P^t(0,S,Q) = 1 - P_{s_i}$ and $P^t(1,S,Q) = P_{s_i}$. To obtain the count histogram in an intermediate sensor node, we can use Equation 2 as long as all child nodes are leaf nodes. The more general task of merging two count histograms where $k > 1$ can be solved by multiplying them *component-wise*.

Consider Figure 3 where s_1, s_2 are intermediate nodes and send their count histograms to root node s_0. We can model their probabilistic distribution (shown in the tables besides the nodes) as polynomials, e.g., for node s_1 we would have $S_1(x) = S_{1,0}+S_{1,1}x+S_{1,2}x^2+\cdots+S_{1,n}x^n$, where each $S_{1,j} = P^t(j,S,Q)$ and a similarly defined polynomial $S_2(x)$ for node s_2. Merging the probabilistic counts of those two nodes into the root node s_0 becomes a matter of simply computing:

$$\begin{aligned} S_0(x) &= S_1(x) \times S_2(x) \\ &= S_{0,0} + S_{0,1}x + S_{0,2}x^2 + \cdots + S_{0,n+m}x^{2n+m}, \end{aligned}$$

from where we can obtain each probabilistic count at node s_0, i.e., $P^t(j,S,Q) = S_{0,j}$

In every node, we can stop the computation at k, because we only need the first k probabilistic counts to answer our queries. Thus, it suffices to implement a simple (component-wise) polynomial multiplication algorithm taking $O(k^2)$ time.

Example 4. *Consider the wireless sensor network in Table 1 at time t = 11:40 and the topology of Figure 2. Let $k = 1$. Sensors s_1, s_3 and s_4 are leaf nodes and send their values to their parent node. s_2 is intermediate node and therefore computes the intermediate probabilistic counts for $k = 0$ and $k = 1$. This means s_2 needs to merge $S_3(x) = 0.3 + 0.7x$ and $S_4(x) = 0.6 + 0.4x$ by multiplying them pairwise and then incorporate its own probability value $P_{s_2} = 0.8$. Again the order is irrelevant. Here we only have two polynomials $S_3(x)$ and $S_4(x)$ resulting in $S_3(x) \times S_4(x) = 0.18 + 0.54x$. Next, we incorporate P_{s_2} and get our final intermediate histogram with $S_2(x) = 0.036 + 0.252x$. Sensor s_2 forwards the probabilistic counts to the root. The sink finally multiplies $S_1(x) = 0.8 + 0.2x$ and $S_2(x) = 0.036 + 0.252x$ to compute the probabilistic counts (= coefficients of resulting polynomial) of the whole tree $S_0(x) = 0.0288 + 0.2088x$.*

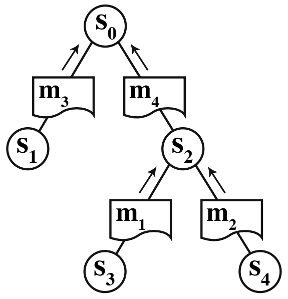
4.4 An Incremental In-Network Algorithm

Next we present an algorithm that brings together both, the in-network aggregation of Section 4.3 and the incremental update strategy of Section 4.2. To enable updates of intermediate count probabilities, every intermediate node has to save its current count histogram as well as the count histograms of its child nodes along with their associated id as unique key. In the first round $t = 0$ we process the query as proposed in Section 4.3 but store all required values in intermediate sensors. In all subsequent rounds $t = t + 1$, only updates are reported and processed as described in the following. We start with the updating process for child count histograms.

There are two types of updates: either a number of old-new value pairs, or a whole updated function. Whenever an intermediate node receives an update, it either updates the child polynomial as described in Section 4.2 or it replaces the whole function with the new function. When $\hat{P}^t(0,S,Q) = 1$ and $\hat{P}^t(j,S,Q) = 0$ for all $(0 \leq j \leq k)$ after removing an old probability value and at the same time no new probability value is sent (sensor either changed its probability to zero or one), we remove the whole count histogram[3]. After processing the updates of the child nodes the intermediate node needs to update its own polynomial. Again we need to consider two cases: We can either merge all updated child polynomials analog to Section 4.2 or use the old-new value pairs of all child nodes to compute the new probabilistic count function. Obviously, we have to multiply all child polynomials as soon as the histogram of a child node was replaced by a new one.

For the number of messages the following applies: If we start with sending updates in the manner of the

3. This usually happens only when leaf nodes are updated.



F = 0.0288 + 0.2088x

Probabilistic Count Histogram of child nodes:

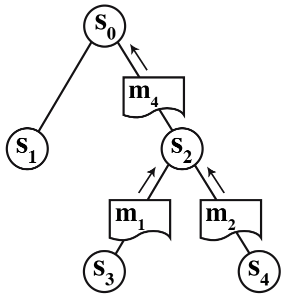
i (key)	Function F_i	
	$c_0 = P^t(0, S, Q)$	$c_1 = P^t(1, S, Q)$
1	0.8	0.2
2	0.036	0.252

F_2 = 0.036 + 0.252x

Probabilistic Count Histogram of child nodes:

i (key)	Function F_i	
	$c_0 = P^t(0, S, Q)$	$c_1 = P^t(1, S, Q)$
3	0.3	0.7
4	0.6	0.4
2*	0.2	0.8

(a) $t =$ 11:40



F = 0.12 + 0.56x and C_1^t=1

Probabilistic Count Histogram of child nodes:

i (key)	Function F_i	
	$c_0 = P^t(0, S, Q)$	$c_1 = P^t(1, S, Q)$
2	0.12	0.56

F_2 = 0.12 + 0.56x and C_1^t=1

Probabilistic Count Histogram of child nodes:

i (key)	Function F_i	
	$c_0 = P^t(0, S, Q)$	$c_1 = P^t(1, S, Q)$
4	0.6	0.4
2*	0.2	0.8

(b) $t =$ 11:50

Fig. 4. In-network Example

incremental centralized algorithm and continue sending only old-new value pairs, the number of messages equals the number of messages sent in approach 4.2. Thus, the number of values increases as we come closer to the root note. However, we want to make sure that analog to the in-network approach, the maximum number of values per message is fixed to $k + 1$. This means we send the whole polynomial as soon as the number of old-new value pairs exceeds $\frac{k}{2}$.

With this, on one hand, we reduce the number of messages if only few sensors report an update, on the other hand we make sure, that in the worst case (all sensors update) the number of messages does not surpass the number of messages sent in the in-network approach.

Example 5. *We process the initialization analog to Example 4 but store all relevant data structures. Figure 4a is a snapshot, illustrating the memory structure at time t = 11:40. At time t+1 =11:50 only s_1 and s_3 change their values: $P^Q_{s_{1,t+1}} = 0.0$ and $P^Q_{s_{3,t+1}} = 1.0$. Sensor s_3 sends an update message with its unique key, old value and in this case no new value but its incremented counter $C^t_1 = 1$ to sensor s_2. Sensor s_2 uses the key to find the matching count histogram and updates it by using Equations 5 and 6:*

$$\hat{P}^t(0, S, Q) = \frac{P^t(0, S, Q)}{1 - P^Q_{s_{1,t}}} = 1$$

$$\hat{P}^t(1, S, Q) = \frac{P^t(1, S, Q) - \hat{P}^t(0, S, Q) \cdot P^Q_{s_{1,t}}}{1 - P^Q_{s_{1,t}}} = 0$$

Since all values but the first one of s_3's count histogram are 0 and the $\hat{P}^t(1, S, Q)$ is 1, we delete its entry in s_2 and increment the counter of s_2 by the value of the sent counter. With s_4 and s_2 keeping their values we only need to update the resulting count histogram in s_2 to complete the update. As no other child node sent an update with a completely new count histogram, we also use Equations 5 and 6 to update the resulting probabilistic counts:

$$\hat{P}^t(0, S, Q) = \frac{P^t(0, S, Q)}{1 - P^Q_{s_{1,t}}} = 0.12$$

$$\hat{P}^t(1, S, Q) = \frac{P^t(1, S, Q) - \hat{P}^t(0, S, Q) \cdot P^Q_{s_{1,t}}}{1 - P^Q_{s_{1,t}}} = 0.56$$

As the number of updated values exceeds the threshold value ($1 > \frac{1}{2}$), we forward the whole polynomial to the intermediate node. The update of sensor s_1 is processed analog to the update of s_3 in s_2 and the entry of s_1 is deleted in s_0. Now, sensor s_2 sends its update with the whole count histogram. Sensor s_0 replaces the whole histogram and increments its counter. To update the histogram in s_0, we now need to merge the child polynomials as described in Section 4.3. But as there is only one entry, we do not need to perform a polynomial multiplication and have our final result. Figure 4b shows the tree after the update was performed.

5 Performance Evaluation

We performed our experiments by varying five parameters: the number of sensors within the network (n), the percentage of uncertain sensors (γ), the probability that a sensors probability value changes from one round to the next round (δ), the probabilistic count (k), and the message size measured in bytes (m). Table 4 shows the values used for those parameters.

The positions of the sensors were randomly chosen within a 100m $\times$ 100m area and each sensor node was assumed to have a fixed wireless radio range of 30m. As mentioned in Section 3, the actual distribution of the probability values is not relevant for the query computation in terms of messages sent – hence not a relevant parameter for our performance evaluation. But for the sake of completeness the probability values follow a normal distribution N(0.5,0.5). Results are based on an average of 10 simulation runs whereas each run consists of 100 time stamps. All generated instances of the WSNs used a hop-wise shortest-path tree as the routing topology. We assume in all experiments that

TABLE 4
Parameter Values Used in the Performance Evaluation
(Default Values Printed in Bold Type)

Parameter	Values
n (Number of Sensors)	100, 500, **1000**, 2500
γ (Ratio of Uncertain Sensors)	25%, 50%, **100%**
δ (Probability of Change)	25%, **50%**, 75%, 100%
k (Number of Coefficients)	1, 5, **10**, 25
m (Message Size in bytes)	64, **128**, 256

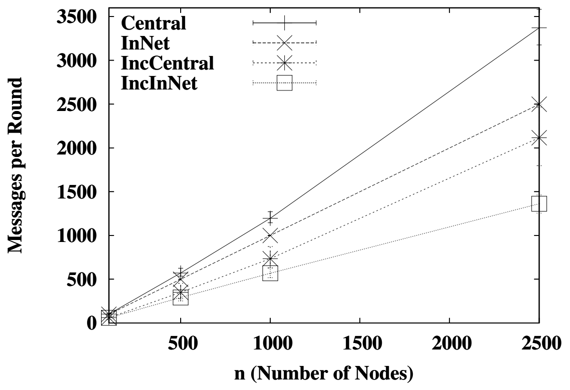


Fig. 5. Simulation Results for n

messages are delivered using a multi-hop setup. Since the query is only sent once from the root to all child nodes and will be amortized over time, we only measure nodes-to-root messages.

Every coefficient was taken into account with a 8 bytes, counters as well as ids were taken into account with 4 bytes each. For the sake of simplicity, we assumed data packages with a header size of 0 bytes.

In the following figures the arithmetic mean is plotted with upper (lower) error bar denoting the overall best (worst) performance. We abbreviate the centralized algorithm with "Central", the centralized incremental algorithm with "IncCentral", the in-network algorithm with "InNet" and the incremental in-network algorithm with "IncInNet".

5.1 Experiments and Results

The results of our experiments are summarized in Figures 5 to 9. In each experiment all parameters but the one in focus are fixed to their default value.

5.1.1 Varying the Number of Sensors n

The foremost trend that we can see in Figure 5 is that IncInNet consistently sends less messages than all others. Both in-network algorithms further improve as the network grows bigger. Particularly in the case of IncInNet the improvement is significant. The reason for this is that the number of values sent per message is restricted to $k + 1$ which is 11 with k set to a default of 10. Thus, with a default maximum payload size of 128 bytes there is no need to send more than one message per sensor. This also explains, why for the InNet approach best case and worst case coincide with the arithmetic mean (no error bars). Since for the InNet algorithm every sensor sends exactly one message per round to its parent node, independent of their respective topology, always $n - 1$ messages are sent[4]. As expected, when n increases, the costs for all algorithms rise as well, however, both in-network algorithms grow slower than the centralized algorithms. Overall, InNet and IncInNet offers better scalability with IncInNet being the overall best solution saving up to 80% of the communication cost compared to Central.

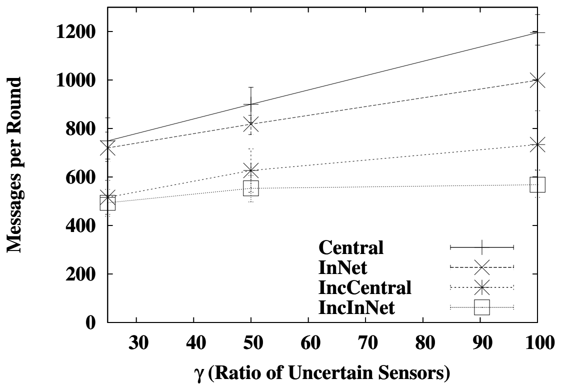


Fig. 6. Simulation Results for γ

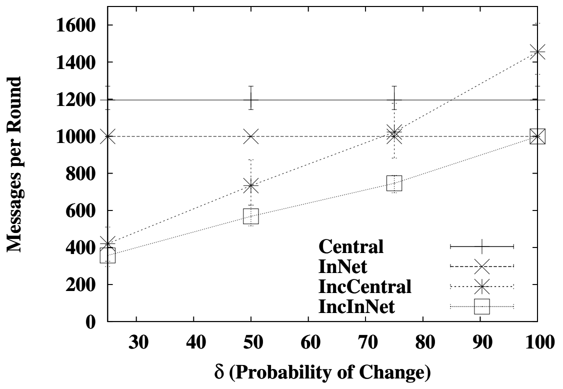


Fig. 7. Simulation Results for δ

5.1.2 Varying the Ratio of Uncertain Sensors

Figure 6 illustrates how a counter as introduced in Section 4.1 affects the performance of the algorithms. Note that for our experiments, the number of certain sensors was equally split into *one* and *zero probabilities*. A counter is taken into account with constantly 4 bytes per message. The larger the number of uncertain sensors the better perform the in-network algorithms. Since in every round different nodes have zero probabilities the error bars become wider.

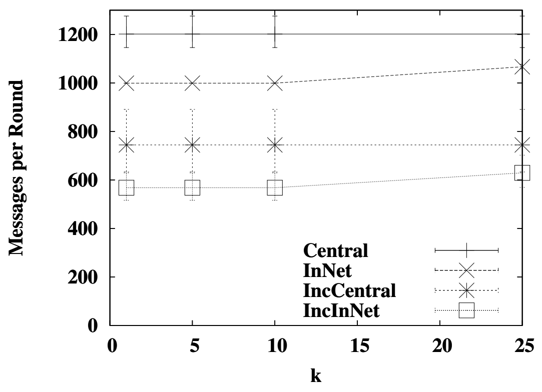


Fig. 8. Simulation Results for k

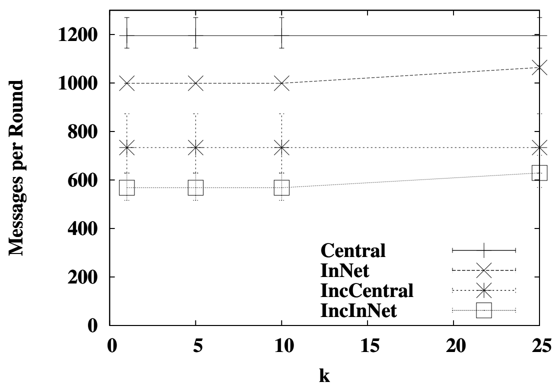


Fig. 9. Simulation Results for m

5.1.3 Varying the Probability of Change

Varying δ creates a scenario that allows observing how the dynamics of the observed probabilities affect the algorithms' performance. Since Central and InNet do not mind updates but always compute the count probabilities from scratch, varying δ does not affect them at all. For IncCentral and IncInNet however naturally applies that the more dynamic the observed values, the more updates will be required. In essence, increasing δ creates more communication traffic in the tree. It is interesting to note that Central outperforms IncCentral for high values ($\delta \geq 75\%$) while IncInNet and InNets performance results even up.

5.1.4 Varying the Count

Figure 8 shows the results of the experiments conducted with varying k. As expected, k only has an influence on InNet and IncInNet. As k increases, the performance of both algorithms slowly decreases. Since we chose k relatively small, varying k does not seem to have a significant effect on the performance at all. Nonetheless, in the extreme case the centralized algorithms outperform the in-network algorithms for $k = n$. This is because InNet as well as IncInNet send $k+1$ values resulting in $n+1$ values (IncInNet additionally also sends a value to identify the sender).

5.1.5 Message Size m

Following [19] we chose (payload) message size of 128 bytes. For the following experiment, m varies within a range of 64 bytes and 256 bytes. Obviously, the smaller the size of a single message, the more messages need to be sent. Figure 9 illustrates the result of our experiments. For Central and IncCentral we therefore observe a steady decreasing number of sent messages. InNet and IncInNet also record a higher number of messages for m =64 bytes. However, the number of messages is constant for m =128 bytes and m =256 bytes, indicating that the total number of total number of bytes never exceeds 128 bytes. Thus, InNet sends $n-1$ messages in total. As mentioned

4. This explanation also applies to the plots that follows.

TABLE 5
Evaluation of Extreme Scenarios

Good Case ($m = 64$, $\delta = 25\%$, $n = 2500$, $\gamma = 100\%$, $k = 1$)			
Central	InNet	IncCentral	IncInNet
4366	2499	1585	835

Bad Case ($m = 256$, $\delta = 100\%$, $n = 100$, $\gamma = 25\%$, $k = 25$)			
Central	InNet	IncCentral	IncInNet
74	74	97	99

in Section 4.1 with unlimited payload size this also applies to Central. In general, with increasing payload size Central draws near to InNet and IncCentral draws near to IncInNet until they finally concide ($m = \infty$).

5.1.6 Relation between n and k

Using our previous observations we estimate that IncInNet performs particularly well in a set up where we choose a small message size, a small probability of change, a high number of nodes, a high ratio of uncertain nodes and a small count. We therefore chose $m = 64$ bytes, $\delta = 25\%$, $n = 2500$, $\gamma = 100\%$ and $k = 1$. In contrast, for the scenario of a bad case we choose a large message size, a high probability of change, a small number of nodes, a small ratio of uncertain sensors and a high count, thus $m = 256$ bytes, $\delta = 100\%$, $n = 100$, $\gamma = 25\%$ and $k = 25$. Table 5 show the results for these scenarios.

As expected, for the optimistic case both incremental algorithms achieve very good results. They reduce the number of messages sent by over 50% compared to the respective non-incremental algorithm. IncInNet is the overall best solution and saves up to 80% to the Central algorithm.

For the pessimistic case the number of uncertain sensors ($\delta \cdot n$) equals k and therefore the in-network algorithms send one more value per message than the centralized algorithms. However, all values can be sent in one message for the non-incremental algorithms. In contrast, the incremental algorithms send a pair of values for every update. Since all values update in each round, they are outperformed by the non-incremental algorithms. IncInNet performs worst since it also needs to send the id with every message.

Generally, we observe that if n is small and k is relatively large the centralized algorithms perform as good as the in-network algorithms whereas the in-network algorithms perform better for a large number of uncertain sensors.

6 Conclusions

In this paper, we studied the problem of answering continuous probabilistic count queries in wireless sensor networks. After formalizing a problem definition we proposed four different algorithms. All algorithms were examined empirically in a performance evaluation. The results show that the incremental in-network algorithm is the overall best solution.

To the best of our knowledge, this paper is the first one that addresses continuous probabilistic count queries in wireless sensor networks. Due to this fact, we opened up new possibilities regarding further studies in different research fields. In this paper we assumed probabilities of two sensors to be mutually independent. One future goal should be finding solutions to cope with more complex uncertainty models where readings of two different sensors can be correlated. Another interesting project would be the transfer of our algorithms to different topologies. By simulating our algorithm on different tree structures we could gain further insight into the strengths and weaknesses of the algorithms. Finally, intermediate count histograms could be used to query subtrees or apply early stopping conditions if a subtree satisfies the query. Future work could address those topics and thereby further reduce transmission costs.

7 Acknowledgements

This research was partially supported by NSERC, Canada and the Academic Exchange Service of Germany (DAAD).

References

[1] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin, "Habitat monitoring with sensor networks," *Commun. ACM*, vol. 47, pp. 34–40, June 2004.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393 – 422, 2002.

[3] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava, "Optimizing sensor networks in the energy-latency-density design space," *IEEE Transactions on Mobile Computing*, vol. 1, pp. 70–80, 2002.

[4] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking queries on uncertain data: a probabilistic threshold approach," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '08, 2008, pp. 673–686.

[5] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: a tiny aggregation service for ad-hoc sensor networks," *SIGOPS Operating Systems Review*, vol. 36, pp. 131–146, December 2002.

[6] R. Cheng, D. V. Kalashnikov, and S. Prabhakar, "Evaluating probabilistic queries over imprecise data," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '03, 2003, pp. 551–562.

[7] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter, "Efficient indexing methods for probabilistic threshold queries over uncertain data," in *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, ser. VLDB '04, 2004, pp. 876–887.

[8] A. Sarma, O. Benjelloun, A. Halevy, and J. Widom, "Working models for uncertain data," in *Data Engineering, 2006. ICDE '06. Proceedings of the 22nd International Conference on*, 2006, pp. 7 – 7.

[9] N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," *The VLDB Journal*, vol. 16, pp. 523–544, October 2007.

[10] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz, "Probabilistic similarity join on uncertain data," in *Database Systems for Advanced Applications, 11th International Conference, DASFAA 2006, Singapore, April 12-15, 2006, Proceedings*, ser. Lecture Notes in Computer Science, 2006, pp. 295–309.

[11] T. Bernecker, H.-P. Kriegel, N. Mamoulis, M. Renz, and A. Züfle, "Scalable probabilistic similarity ranking in uncertain databases," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 9, pp. 1234–1246, 2010.

[12] J. Li, B. Saha, and A. Deshpande, "A unified approach to ranking in probabilistic databases," *Proc. VLDB Endow.*, vol. 2, pp. 502–513, August 2009.

[13] R. Ross, V. S. Subrahmanian, and J. Grant, "Aggregate operators in probabilistic databases," *J. ACM*, vol. 52, pp. 54–101, January 2005.

[14] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang, "Top-k query processing in uncertain databases," in *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, April 15-20, 2007, The Marmara Hotel, Istanbul, Turkey*, 2007, pp. 896–905.

[15] K. Yi, F. Li, G. Kollios, and D. Srivastava, "Efficient processing of top-k queries in uncertain databases," in *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, 2008, pp. 1406–1408.

[16] G. Cormode, F. Li, and K. Yi, "Semantics of ranking queries for probabilistic data and expected ranks," in *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, 2009, pp. 305–316.

[17] B. Malhotra, M. A. Nascimento, and I. Nikolaidis, "Exact top-k queries in wireless sensor networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 99, no. PrePrints, 2010.

[18] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 2002, pp. 1567 – 1576 vol.3.

[19] E. Pinedo-Frausto and J. Garcia-Macias, "An experimental analysis of zigbee networks," in *Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on*, 2008, pp. 723 –729.

[20] S. Wang, G. Wang, X. Gao, and Z. Tan, "Frequent items computation over uncertain wireless sensor network," *Hybrid Intelligent Systems, International Conference on*, vol. 2, pp. 223–228, 2009.

[21] S. A. Kripke, "Semantical analysis of modal logic i normal modal propositional calculi," *Mathematical Logic Quaterly*, vol. 9, pp. 67–96, 1963.

[22] L. Antova, C. Koch, and D. Olteanu, "10 worlds and beyond: efficient representation and processing of incomplete information," *The VLDB Journal*, vol. 18, pp. 1021–1040, 2009.

[23] K. Lange, *Numerical analysis for statisticians.* Springer, 1999.