Fault-Tolerant Queries over Sensor Data

Iosif Lazaridis[1] Qi Han[2] Sharad Mehrotra[1] Nalini Venkatasubramanian[1]

[1]University of California, Irvine [2]Colorado School of Mines
{iosif, sharad, nalini}@ics.uci.edu, qhan@mines.edu

Abstract

We consider the problem of evaluating continuous selection queries over sensor-generated values in the presence of faults. Small sensors are fragile, have finite energy and memory, and communicate over a lossy medium; hence, tuples produced by them may not reach the querying node, resulting in an incomplete and ambiguous answer, as any of the non-reporting sensors may have produced a tuple which was lost. We develop a protocol, FAult Tolerant Evaluation of Continuous Selection Queries (FATE-CSQ), which guarantees a user-requested level of quality in an efficient manner. When many faults occur, this may not be achievable; in that case, we aim for the best possible answer, under the query's time constraints. FATE-CSQ is designed to be resilient to different kinds of failures. Our design decisions are based on an analytical model of different fault tolerance strategies based on feedback and retransmission. Additionally, we demonstrate the good performance of FATE-CSQ compared to competing protocols with realistic simulation parameters and under a variety of conditions.

1 Introduction

Wireless sensors are becoming smaller and cheaper, while their capabilities continuously improve. Systems incorporating large numbers of them are now technically and economically feasible and will provide unprecedented access to the physical world at a fine level of spatio-temporal detail [5]. It is important that data management architectures take into account both the capabilities and limitations of sensors [14]. Increasingly, researchers are realizing that sensors are more than passive beacons, but can perform useful work, to conserve their own resources and to meet application goals. Sensors can be smart in terms of data acquisition [17], processing before transmission [11], and propagation across the network [4]. Pushing intelligent decision-making and data manipulation to sensors has many benefits: (i) removing part of the burden from the centralized components of the system, (ii) making decisions close to the monitored phenomenon, thus improving reaction times, (iii) preserving the crucial resources of wireless bandwidth and energy supply.

International Conference on Management of Data
COMAD 2006, Delhi, India, December 14–16, 2006

Not only performance, but also *semantic* aspects of sensor-based data management must be addressed. In traditional databases, the focus is on storing and retrieving data efficiently: lower-level components such as hard disks, networks, operating systems, etc., are assumed to run correctly and provide reliable service, failing only exceptionally. This is not reasonable for sensor networks where a wide variety of accidents may occur: sensors are fragile units deployed in the field, communication failures, energy depletion and other calamities are routine, and the system must function in spite of them. More importantly, they introduce an aspect of uncertainty into standard operations, such as answering queries. We should not aim to extract some data from the network in a purely best-effort manner, but rather to produce results with a clearly defined formal meaning.

In our paper we try to meet this challenge for the evaluation of *continuous selection queries* (CSQs) over sensor-generated data. We define such queries as requests for the retrieval, for every *period* of length τ, of all sensor values satisfying a user-defined predicate λ. The query is issued at an *injection point* IP, and spreads to a set $\mathcal{S}$ of all sensors of interest. Every τ seconds, each $s \in \mathcal{S}$ generates a tuple t_s by acquiring the attribute(s) of interest, and if $\lambda(t_s)$ is TRUE, then t_s is called a YES tuple which must be forwarded to IP. Otherwise, it is a NO tuple and does not need to be forwarded. The *exact set* $\mathcal{E}_{\lambda,\mathcal{S}}$ of such a query for a particular period is defined as the set of all YES tuples, or $\mathcal{E}_{\lambda,\mathcal{S}} = \{t_s|\lambda(t_s) \wedge s \in \mathcal{S}\}$. We will call this set $\mathcal{E}$, where $\lambda, \mathcal{S}$ are assumed. Due to faults (e.g., a message being lost en route to IP), we expect IP to receive an *answer set* $\mathcal{A}$ that is a subset of $\mathcal{E}$.

The presence of faults introduces two obstacles in interpreting $\mathcal{A}$. Is it a "good" answer, or is it very incomplete due to the occurrence of many faults? Even if $\mathcal{A}$ was "perfect" (equal to $\mathcal{E}$), ambiguity remains: what about the remaining sensors, $|\mathcal{S}| - |\mathcal{A}|$ in number, from which tuples were not received during this period? If faults occur frequently, then it is possible for *all* these sensors to have actually produced results, which however were not reported.

As a practical example, consider the following scenario: a disaster occurs, involving e.g., a leak of poisonous gas. To gain information about the extent and intensity of the disaster, without endangering personnel, small wireless electronic sensors may be dropped from the air covering a wide area around the reported sources of the chemical pollutants. These sensors subsequently report whether or not the pollutant levels are above a critical threshold, implying e.g., that protective gear should be worn by rescue personnel deployed in the area. Diffusion of pollutants may depend on weather (e.g., winds), or on the rate at which they are released to the environment. An ad-hoc sensor network would be able to deliver critical real-time identification of dangerous regions, helping to prioritize resources for rescue operations, determining evacuation urgency in different areas, etc. The presence of faults in this example would imply that locations not reporting "dangerous" levels of pollutants may indeed be dangerous. If decision makers could know that the number of such locations is small, then they would be able to make better decisions, e.g., assessing the risk of sending personnel to a non-reporting region.

Our paper deals with how to give to the injection point IP additional *quality guarantees* about the answer set $\mathcal{A}$. The query has to specify its *quality requirements*, expressing how "good" it wants $\mathcal{A}$ to be. Then, the system will work towards meeting these requirements. Sometimes, due to the occurrence of many faults, it might be impossible to achieve this goal; then the best possible answer will be given within the period time τ. Irrespective of whether or not the requirement is met, the system will produce guarantees, aiding in the interpretability of the result.

Our paper is structured as follows. Section 2 presents challenges in dealing with faults in sensor networks and defines a metric for gauging the goodness of a query answer, In Section 3 we present a protocol, FAult Tolerant Evaluation of Continuous Selection Queries (FATE-CSQ) for producing such an answer. FATE-CSQ is based on hop-based feedback and re-transmission, a design decision motivated by our analytical study of alternative fault tolerance mechanisms presented in Section 4. We evaluate FATE-CSQ and alternatives in Section 5. Finally, we review some related work in Section 6 and conclude in Section 7 presenting directions of future research.

2 Faults in Sensor Networks

In this section, we describe our problem setting, and a metric for quantifying answer quality in the presence of faults.

2.1 Problem Setting

We assume that a sensor network consists of *nodes*, which can be wireless sensors, wired access points or servers. Nodes communicate with each other via pairwise *links*. Each sensor has a link with every other sensor within its hearing range. A link is a logical concept, capturing the ability of two nodes to talk to each other without intermediaries. When a node s transmits, then its message can be heard by all nodes sharing a link with s: this may lead to collisions, but can also be used for reaching many nodes with one transmission, saving energy. The query is posed at IP and answer tuples must flow from all nodes in $\mathcal{S}$ to IP; this imposes some structure to the part of the sensor network we are dealing with, IP is a "head" node, and other nodes have paths through which tuples can flow to IP.

We categorize faults along two dimensions: node vs. link faults and terminal vs. non-terminal ones. A *node* fault is a failure of a component A in the system and affects all its *descendants*, i.e., nodes relying on A to transmit data to IP. A *link* fault occurs when a transmitted message is lost, even though both endpoints are operational, because of e.g., collisions, or electromagnetic interference [30].

A second distinction pertains to fault *finality.* Some faults are caused by conditions which cannot be circumvented by additional effort or a change of policy, e.g., physical damage to a node. Effort towards correcting such *terminal* faults is not useful. By contrast, *non-terminal* faults, e.g., reception of a corrupt packet, can be corrected, leading to the recovery of data and improving the probability that they will reach IP. Such faults can be further categorized based on their *duration. Short-term* faults may cease to be a problem *before* the beginning of the next period, in time for the data to be forwarded to the user. *Long-term* faults last for more than one, and potentially many, periods; hence, effort towards resolving them during the current period is wasteful.

Our protocol, FATE-CSQ aims to prevent, detect, and correct faults. It prevents faults by occasionally modifying the network topology to ensure that healthy nodes always have a path towards IP. It detects faults using an intelligent feedback-based mechanism. Finally, it tries to correct faults by attempting to retransmit data that have not been properly forwarded.

2.2 Quality Metric

To assess the quality of the answer, we have chosen to use the *recall* measure, which has a long history of use in Information Retrieval [2], and has also been recently proposed [15] for dealing with imprecise data. Recall can be computed easily and has an intuitive real-world meaning. If $\mathcal{A}$ is the answer set, a subset of the exact set $\mathcal{E}$, then recall is simply the fraction of $\mathcal{E}$ that has been retrieved:

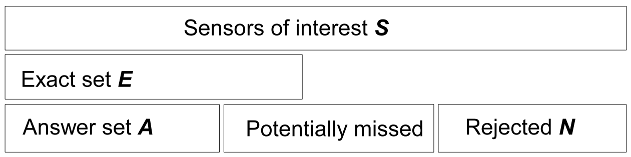

Figure 1: Sensor set $\mathcal{S}$, exact set $\mathcal{E}$, answer set $\mathcal{A}$, number of No tuples N.

$$r = \frac{|\mathcal{A}|}{|\mathcal{E}|} \text{ if } |\mathcal{E}| \neq 0 \text{ , else } r = 1 \tag{1}$$

$|\mathcal{A}|$ is known: it is the number of answer tuples received by IP, but $|\mathcal{E}|$ is unknown and can vary in $[|\mathcal{A}|, |\mathcal{S}|]$. For example, if $|\mathcal{A}| = 100$ and $|\mathcal{S}| = 1000$, then r can be as low as $\frac{100}{1000} = 0.1$, since potentially all 1000 tuples were Yes, but only 100 were received due to faults. Without additional information, this is the best we can do. We can improve this bound, if, in addition to $\mathcal{A}$, the IP is also aware of a number N of tuples known to be No. If N is known—the mechanics of how to achieve this are explained in the next section—then we can improve on the recall guarantee: now, at most $|\mathcal{S}| - N$ tuples may exist in the exact set. In the example of the preceding paragraph, if we knew that $N = 500$ tuples were No, then r can be as low as $\frac{100}{1000-500} = 0.2$, a definite improvement over the previous bound. A graphical representation of the recall concept is seen in Figure 1.

In general, the *recall guarantee* r_g is:

$$r_g = \frac{|\mathcal{A}|}{|\mathcal{S}| - N} \text{ if } |\mathcal{S}| \neq N, \text{ else } r_g = 1 \tag{2}$$

Summarizing, a query q is applied on a set of nodes $\mathcal{S}$ and consists of a predicate λ, a period τ, and a recall requirement r_q. Its semantics are: at time t the set of tuples satisfying λ is $\mathcal{E}$. By time $t + \tau$ a subset $\mathcal{A} \subseteq \mathcal{E}$ should reach IP, and a count N of sensors in $\mathcal{S}$ whose tuples are No. $\mathcal{A}$ and N should be such that $r_g \geq r_q$ (where r_g is defined from Eq. 2); if this is not achievable, then work towards minimizing $r_q - r_g$ until time $t + \tau$. This is repeated for the next period, setting $t \leftarrow t + \tau$.

3 FATE-CSQ Protocol

In this section we will describe the three phases of operation, shown in Figure 2, of FATE-CSQ. We will be using Figure 3 to illustrate the workings of our protocol.

3.1 Query Establishment Phase

The query is first submitted to IP and during the first phase of operation, it must reach all nodes in $\mathcal{S}$. This is achieved by establishing a routing tree using broadcasts. Each node s maintains its parent's ID,

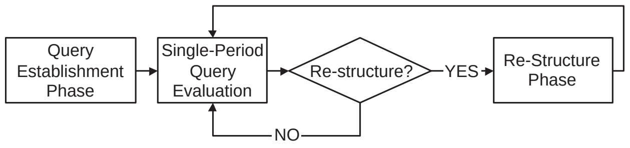

Figure 2: Phases of FATE-CSQ Operation

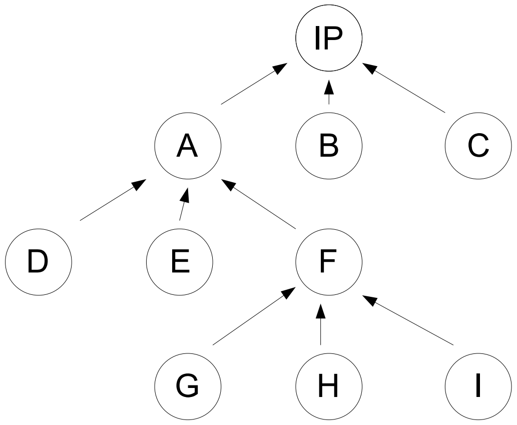

Figure 3: Example of a sensor network

P_s: this is the node from which it has received the broadcast, e.g., $P_F = A$. Subsequently, each node sends messages to IP via its parent, e.g., node I sends a message to IP via the path $I \rightarrow F \rightarrow A \rightarrow IP$. Each parent s maintains a set $\mathcal{C}_s$ of its children.

The procedure described in the previous paragraph can be repeated: nodes who already have a parent retain it, but nodes who have missed the initial message are given an opportunity to join in, just as before. After a few repetitions, the full routing tree such as in Figure 3 is established. IP counts n_{IP}, the number of sensors reporting via the tree, and $|\mathcal{S}|$ is set to be equal to n_{IP}; this may be smaller than the total number of nodes, because some of them may not have joined the tree despite repeated attempts. Alternatively, $|\mathcal{S}|$ can be set as the total number of deployed nodes. In that case $|\mathcal{S}| - n_{IP}$ tuples will always be potentially missed. The choice of $|\mathcal{S}|$ is a matter of desired semantics, i.e., recall over all nodes, or over all nodes currently connected in the routing tree.

The beginning time t_0 of the next phase, i.e., the start of the first period of the query evaluation, is then announced to all nodes via the routing tree. At this time, the round length is also announced; the concept of a round will be defined below. We assume that clocks of different nodes are synchronized, i.e., that t_0 refers approximately to the same instant for all sensors in the system. Recent work [9] indicates that fairly close synchronization (< 1 msec) is feasible.

3.2 Single-Period Query Evaluation Phase

This is the main phase of the FATE-CSQ protocol, repeated for each period. The answer is progressively improved in a series of smaller time units, called rounds, during which tuples flow towards the IP and are acknowledged by their recipients with feedback messages. Nodes finish work when the recall requirement is met, or the current period expires, or when they exhaust all possible work that can be done in their local subtree, and hence can go to sleep.

The first period begins at time t_0. Each sensor s generates its tuple t_s by sensing its physical environment at that time. This phase lasts until $t_0 + \tau$ at the latest, as the system tries to produce results that meet r_q. There are three different kinds of nodes: (i) leaf nodes (e.g., G, C) are responsible only for transmitting their own values to their parent, (ii) intermediate nodes (e.g., A, F) are additionally responsible for transmitting the values of more distant descendants, (iii) IP does not need to transmit any values, but is responsible for determining when r_q has been met.

Transmission Rounds.— Transmission of data and feedback within a period is organized in successive *rounds*. Each round is a time interval during which a parent receives data from its children: this has to be enough to accommodate all of them and is thus longer for nodes with many children. Unlike τ which is query-specific, a round length is specific to each parent in the network and is known by each of his children. To avoid collisions, children must not all broadcast simultaneously. Elaborate policies, such as TDMA [12] can be used to allocate time for transmission to each child: any such policy can be used with FATE-CSQ, as long as transmission is contained within the duration of the round.

There are two different kinds of data transmitted from child to parent: (i) YES tuples, and (ii) counts of NO tuples. YES tuples need to be sent individually, because they will be added to the set $\mathcal{A}$ returned to the user. For a node s, the number N_s is the count of NO tuples that s knows to exist in its subtree: this is less or equal to the actual number of NO tuples—since some reports of NO tuples may be lost. Unlike YES tuples which must reach IP individually, counts of NO tuples can be aggregated. A node's N_s is always at most $\sum_{z \in \mathcal{C}_s} N_z$, the sum of its children's counts. Data transmission in FATE-CSQ occurs as follows:

Transmitting Data:
(1) Send your own tuple if YES; otherwise send a NO message to your parent.
(2) Listen for messages from your children.
(3) If you receive a YES tuple, forward it to your parent.
(4) If you receive a new count N'_z from child z, then update your own count as $N_s \leftarrow N_s + N'_z - N_z$ and forward the new N_s to your parent.

All nodes in the system run the above, but leaf nodes do not perform steps (2-4) and IP outputs YES tuples to the user. IP checks whether $r_g \geq r_q$ and if so, emits a STOP message to all nodes in $\mathcal{S}$, halting work until the next period. Step (1) occurs only if necessary, as we will see next.

Feedback.— At the end of the round, the parent transmits feedback to its children about their own tuples received during the round. For example, node A transmits feedback about tuples t_D, t_E, t_F. This is called *direct feedback* and uses an entity called *Children Feedback Vector* (CFV).

Forwarding feedback is used for tuples forwarded by a node's children. This uses an entity called *Missing Tuple Vector* (MTV). We will explain CFV and MTV shortly, but note that they apply to different sets of tuples: if t_F is lost in link $F \rightarrow A$, then this will be corrected by A using direct feedback (CFV), because F is a child of A. If t_H is lost in the same link, then forwarding feedback (MTV) will be used, because H is a more distant descendant of A.

In direct feedback, each s has either received the value of each of its children or not. The CFV is a $|\mathcal{C}_s|$-long bit vector with 0's for sensors whose tuples (either YES or NO) have been received and 1's otherwise. At the end of the round, the CFV is broadcast. Children hear it and do the following:

Direct Feedback:
(1) Listen for the CFV at the end of each round
(2) On hearing the CFV, check whether your bit is set. If yes, re-transmit your tuple; else, do nothing.

Using the CFV has three advantages: (i) feedback is given in a single message for all children, (ii) each child's tuple is relayed at most once to its parent,[1] and (iii) feedback is timed to coincide with the end of the round; hence, no idle listening on the channel is done to receive it. Additionally the CFV can be used to set the duration of the next round: e.g., CFV=011000 has two 1's and hence the next round should last $\frac{2}{6} = \frac{1}{3}$ times the length of the first round, since now 2 instead of 6 children need to transmit data. The transmit-feedback cycle becomes faster as more data are received by the parent, improving latency, and increasing the probability that the quality requirement will be met. If a node does not receive the CFV then it will be stuck listening in (1) of the above, and pick the CFV in the next or subsequent rounds. The CFV is small, so it might be attached to all messages generated by a node's siblings: this allows a node to receive it indirectly, reducing idle listening time.

Let's proceed to forwarding feedback. We treat this separately, because nodes forward tuples from their entire subtree, and this can be huge, especially for the higher-level nodes, close to IP. We would have to reserve a bit for every such node if we used a CFV vector to provide feedback, thus increasing the CFV vector to a prohibitively large length; moreover, this

[1]This corresponds to step (1) of the protocol for Transmitting Data. Of course, a tuple may be retransmitted more than once if the CFV itself is lost.

would require topological knowledge about each node's entire subtree. Node s forwards a sequence of tuples to its parent P_s during a round. It numbers these sequentially, e.g., $(1, 2, 3, 4, 5, 6)$. The highest sequence number is labeled $n_{max} = 6$. Node s also forwards N_s to its parent whenever this changes;[2] suppose $N_s = 5$ for our example. Now, suppose that P_s receives tuples numbered $(2, 3, 5)$ and the latest count of NO tuples it has received is N_s^{old}. The MTV which it supplies to s at the end of the round will consist of: (i) the highest tuple number received n, in our example $n = 5$, (ii) the missing values encoded in a bit vector with 1's representing gaps and 0's received tuples. Thus $(2, 3, 5)$ is represented as 10010 (read left to right), (iii) the most recent count of NO tuples N_s^{old}; suppose $N_s^{old} = 3$ in our example. When s receives the MTV, then it will:

Forwarding Feedback:
(1) Resend tuples with seq. number $\leq n$ whose bit in the MTV is 1.
(2) Resend tuples with seq. number $n + 1$ to n_{max}.
(3) Drop all other tuples from its buffer.
(4) If $N_s^{old} < N_s$ send N_s.

In our example, tuples numbered $1, 4$ would be retransmitted in step (1), tuple 6 in step (2) and $N_s = 5$ (since $N_s = 5 > N_s^{old} = 3$) in step (4). Tuples $(2, 3, 5)$ would be dropped from the buffer of s.

Forwarding feedback is used to isolate the effects of faults. If a tuple is lost in a certain link, it need not be re-transmitted from the node where it originated: this improves both latency, as faults are corrected quickly, and reduces communication load. As an additional benefit, parts of the routing tree can go to sleep once a certain condition has been met, as we will explain below.

Preventing Buffer Overflows.— There is a potential danger, stemming from the finite memory buffer of a sensor, and the fact that it must forward all answer tuples from its subtree to its parent. In the worst case, all descendants of a sensor s are YES and there is a fault either of P_s or of the link $s \rightarrow P_s$. In that case, all these (n_s in number) tuples will be "stalled" at s, since they are not acknowledged by P_s. This may cause a buffer overflow, and hence a lost tuple. To solve this problem, each sensor s keeps track of its buffer, checking, when receiving a new tuple, whether or not it can be stored locally. Only if this is possible does it acknowledge the receipt via feedback. Thus, the children of s do not delete a tuple if there is no room in s to store it. Under this policy the problem of lost tuples, as in the previous example would be solved: when normal operation in link $A \rightarrow IP$ is restored, then A's buffer will again have space, and A can start acknowledging received tuples once more.

[2] N_s can be piggy-backed on other packets transmitted to one's parent, e.g., YES tuples to save communication effort.

Going to Sleep.— Parts of the network must be allowed to go to sleep when they have produced all possible information. We observe that each node s knows the number n_s, the number of nodes in its subtree; this can be easily determined during the Query Establishment Phase by counting messages flowing from each node to the IP. Node s can also count Y_s, the number of tuples that it has forwarded and have been acknowledged by its parent: this can be done by incrementing Y_s whenever a forwarded tuple is dropped from the sensor's buffer as a result of forwarding feedback from its parent. When s receives feedback from P_s then it will perform the following check: $N_s^{old} + Y_s = n_s$ which implies that P_s received all tuples forwarded by s and also has the correct count of NO tuples. Thus, s can now go to sleep for the rest of the period.

3.3 Re-Structure Phase

The STOP signal marks the end of the Query Evaluation phase for the current period. If STOP is not transmitted, then the system has failed to meet the recall requirement, and the Query Evaluation phase ends automatically. The decision whether or not to re-structure must take place before the next period begins. The initial network topology, discovered in the Query Establishment phase will slowly become outdated. As nodes fail, their descendants will be cut off from IP and large parts of the network will become "silent." It will be increasingly difficult (if at all possible) to meet r_q, taking a greater fraction of τ, as nodes perform repeated rounds trying to squeeze as much data as possible out of the still-connected parts of the network.

Thus, we must occasionally initiate a Re-Structure phase, during which, continuous query evaluation is interrupted. IP sends a RESTRUCTURE message: parent-child information is modified, so that any nodes whose communication with their parent is problematic acquire a new parent and can start producing data again. The Re-Structure phase proceeds similarly to the Query Establishment phase, but it is now not necessary to re-transmit the query itself, or to re-establish the starting time of each period. The Re-Structure phase is an overhead: doing it frequently keeps the network in good shape, but wastes time and energy as network topology is re-established; conversely, doing it rarely decreases the overhead, but the network deteriorates. As we will show in Section 5, there is an optimum rate at which re-structuring ought to take place.

4 Analysis of Mechanisms for Fault-Tolerance

FATE-CSQ relies on a local, hop-based (HOP) feedback mechanism to provide resilience to faults. Alternative methods, use either (i) end-to-end (E2E) feed-

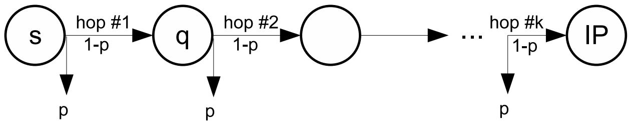

Figure 4: A path from node s to IP. Reverse links (for feedback) are not shown.

back between the IP and the sensor producing a tuple, or (ii) optimistically (OPT) no feedback at all. E2E methods are "forward-and-forget" and do not require intermediate nodes to take any special action. OPT mechanisms save on the cost of sending the feedback.

In this section, we analyze the HOP, E2E and OPT methods. We will focus on a node s which is k hops away from the IP (see Figure 4). For simplicity, we assume that feedback is given individually, rather than combined for multiple nodes as in FATE-CSQ. This overestimates the number of feedback messages, but suffices for building our intuition. Additionally, we do not deal with No counts which represent a minor cost compared to Yes tuples and can usually be piggy-backed on such tuples.

Let p be the failure rate (probability) between a pair of nodes. A message can fail either because the recipient or the link between sender and recipient are faulty, and p combines both factors. We will estimate the number of messages (data or feedback) sent. This is related to energy expenditure, as well as time: more messages will drain energy resources more rapidly, and will require more time.[3]

Optimistic Protocol - OPT.— In OPT, a node forwards tuples to its parent. It does not provide any feedback to its children. To increase the probability of reception, the source node attempts to send each message m times The LAZY protocol discussed in Section 5 is a special case of OPT with $m = 1$.

A tuple will not reach the IP if there exists at least one hop for which it fails. Hence, in a single attempt it will reach the IP with probability $P_{k,1}^{OPT} = (1-p)^k$. For m attempts, it will reach the IP with probability:

$$P_{k,m}^{OPT} = 1 - (1 - (1-p)^k)^m \tag{3}$$

$P_{k,m}^{OPT}$ increases as p decreases (less faults), k decreases (less hops), and m increases (more retries).

The expected number of messages (for all nodes in the path) is calculated as follows. For one try, the expected number of messages, if the node is k hops away from the IP is:

$$M_{k,1}^{OPT} = 1 + (1-p)M_{k-1,1}^{OPT} \text{ with } M_{0,1}^{OPT} = 0 \tag{4}$$

[3]The relationship is not, however, entirely linear. For example, a message may take longer to reach a parent with many children (longer round size).

Solving this recurrence yields $M_{k,1}^{OPT} = \frac{1-(1-p)^k}{p}$, with a special case of $M_{k,1}^{OPT} = k$ for $p = 0$. Since the value is sent m times, the expected number of messages is:

$$M_{k,m}^{OPT} = m\frac{1-(1-p)^k}{p} \tag{5}$$

This increases with more retries m, lower probability of failure p and greater number of hops k.

End-to-End Protocol - E2E.— In E2E the IP sends feedback to the sensor s. Intermediate nodes only relay the feedback. If the message is received by IP, positive feedback is provided; otherwise, negative feedback is sent. If the sensor s receives positive feedback then it takes no further action; if it receives negative or no feedback, then it retransmits its tuple. Re-transmissions always begin from the source s.

During a single try, the value will reach IP with probability $P_{E2E,1} = (1-p)^k$. Regardless of whether the value reaches the IP or not, feedback (positive or negative) will be generated which will reach s with the same $P_{E2E,1}$. Hence, with probability $P_{E2E,1}^2 = (1-p)^{2k}$ node s will receive positive feedback, and hence no more messages will be sent. With probability $1-(1-p)^{2k}$ it will either receive no or negative feedback and the tuple will be resent. If m end-to-end transmissions are attempted in total, then the tuple will reach IP with probability $P_{k,m}^{E2E} = 1-(1-P_{E2E,1})^m = 1-(1-(1-p)^k)^m$ just as in OPT.

The expected number of messages of this protocol is derived as follows. Sending either the data value or the feedback takes $M_1 = \frac{1-(1-p)^k}{p}$ messages in the expected sense for one attempt. The expected number of messages, if m total number of attempts are allowed is $M_{k,m}^{E2E} = 2M_1 + (1-P_{E2E,1}^2)M_{k,m-1}^{E2E}$ where $M_{k,1}^{E2E} = M_1$. Note that $2M_1$ refers to the messages exchanged in the first end-to-end attempt. Then, with probability $1-P_{E2E,1}^2$ either the message from s to IP or the feedback will be lost, resulting in a second attempt. $M_{k,1}^{E2E}$, the final term is simply M_1, because no feedback is needed in the very last attempt: the node will not retransmit irrespective of whether its tuple has been received by the IP or not. Solving the recurrence, yields:

$$M_{k,m}^{E2E} = \frac{M_1(2 + (1-P_{E2E,1}^2)^{m-1}(-2+P_{E2E,1}^2))}{P_{E2E,1}^2} \tag{6}$$

The number of messages increases as $P_{E2E,1}$ decreases (frequent losses), and as m increases (more attempts).

Hop-by-hop Protocol - HOP.— This is used by FATE-CSQ. Positive or negative feedback is given at each individual hop. A node retransmits to its parent unless positive feedback is received.

Consider a single hop. The probability of a message being delivered to q (one hop) is $P_{1,m}^{HOP} = 1 - p^m$, if

m attempts are made. In general we can write the probability that the message will reach the IP if m attempts are allowed in the first hop and there are k hops in total as:

$$P_{k,m}^{HOP} = \begin{cases} 1 - p^m & \text{if } k = 1 \wedge m > 0 \\ 0 & \text{if } m = 0 \\ (1-p)P_{k-1,m}^{HOP} + pP_{k,m-1}^{HOP} & \text{else} \end{cases} \tag{7}$$

The first case is a boundary condition if the message is only one hop away from the IP. If $m = 0$ (second case) then the message cannot be delivered, as there are no more retries left. Finally, for the third case: if the message is successfully forwarded by one hop, then it must clear $k-1$ hops with m retries;[4] otherwise it must clear all k hops with one less $(m-1)$ number of retries. Now, consider the expected number of messages for m tries:

$$M_{k,m}^{HOP} = \begin{cases} 0 & \text{if } m = 0 \text{ or } k = 0 \\ 1 + AM_{k,m-1}^{HOP} + BM_{k-1,m}^{HOP} & \text{if } m = 1 \text{ and } k > 0 \\ 2 + AM_{k,m-1}^{HOP} + BM_{k-1,m}^{HOP} & \text{else} \end{cases} \tag{8}$$

If $m = 0$ (no more attempts) or $k = 0$ (no more hops), then no more messages are sent. Otherwise, 2 messages are sent (tuple transmission, feedback), except in the final attempt, where no feedback is necessary. More messages will be sent in the first hop itself if either the tuple transmission or the feedback failed (probability $A = 1 - (1-p)^2$), but now the number of retries is reduced by one (second term). Finally, we add up the number of messages further up in the path to the IP, i.e., starting from $k-1$ hops away from the IP. Such nodes produce work only with probability $B = 1 - p$, i.e., if the message was successfully delivered in this hop.

4.1 Which protocol is best?

We can now analytically determine which approach should be chosen, depending on p, the probability of failure. Notice that we calculated the probability P of the IP receiving the value for the three different protocols. This corresponds exactly to recall r_q: for each protocol. We would like to set m to a value which reaches r_q level of probability. Subsequently, using this m we can calculate the expected number of messages M: the best protocol is the one which minimizes this M. Note the qualitative difference between OPT and the others: OPT always performs m attempts, where m must be set a priori: if p is lower than expected, then more work than necessary is performed; coversely, if p is higher, then r_q is not achieved. This is not a problem for E2E and HOP, which stop exactly when m suffices to achieve r_q.

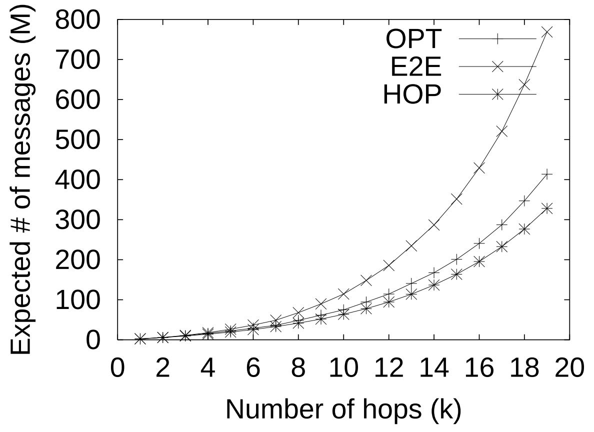

Figure 5: Number of messages vs. number of hops

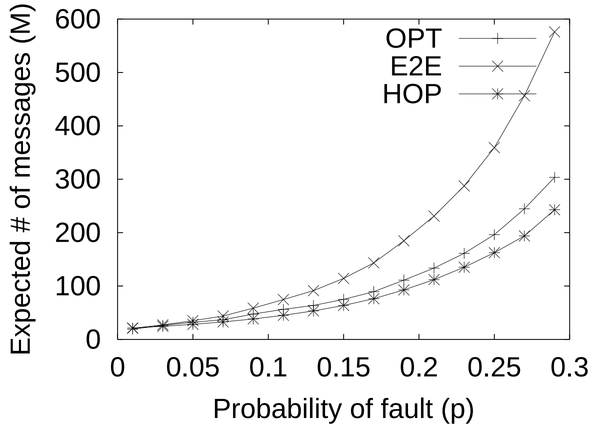

Figure 6: Number of messages vs. failure rate

Solving $r_q \geq P$ from equation 3 yields optimal value $m_{OPT} = m_{E2E} = \frac{\log(1-r_q)}{\log(1-(1-p)^k)}$ for OPT and E2E. For HOP this is determined by calculating $P_{k,1}^{HOP}, P_{k,2}^{HOP}, \ldots, P_{k,i}^{HOP}$ from recurrence equation (7) stopping when $P_{k,i}^{HOP} \geq r_q$; then $m_{HOP} = i$. Substituting $m_{OPT}, m_{E2E}, m_{HOP}$ in Eqs. 5,6,8 we obtain the cost incurred by the three protocols.

To illustrate the behavior predicted by our analytical model, we plot the expected number of messages as a function of k (Figure 5 with $p = 0.15$) and p (Figure 6 with $k = 10$); we keep the recall requirement at $r_q = 0.95$. The better performance of the hop-by-hop method is illustrated; note, that in FATE-CSQ performance will be likely even better, because messages are not acknowledged individually.

[4]The justification for using m is the following: the message must clear one less hop ($k-1$ vs. k). However, transmitting the value in the first hop itself consumes one message time. Hence, the system has enough time to afford the same number of retries for q as for s.

5 Performance Study

We now validate FATE-CSQ, and test its performance under different conditions.

5.1 Simulation Settings

There are no published algorithms providing recall guarantees for evaluating continuous queries. Therefore, we compare FATE-CSQ against two heuristics:

LAZY.— This is OPT of Section 4 with $m = 1$. Given that LAZY does not take faults into account, its performance is purely dependent on the status of the sensor network and thus serves as a baseline for comparison: it will do the minimum amount of work possible, which will only suffice to meet r_q if faults are sufficiently rare.

E2E.— This is E2E described in Section 4. Additionally, feedback messages sent to multiple nodes are combined. Since the feedback is initiated by the IP, its size can be very big. In order to break it into smaller pieces, the IP assigns a logical ID for each node, in essence transforming the routing tree into an index on these logical IDs. In this way, intermediate nodes decrease the feedback message size by only including those nodes whose logical ID belongs to current subtree in the routing tree.

To compare policies we use: (i) the guarantee r_g; (ii) latency (time to achieve r_q); and (iii) normalized energy consumption, i.e., total energy consumption divided by $|\mathcal{S}|$.

We simulated all the three protocols (LAZY, FATE-CSQ and E2E) on GlomoSim [29], a scalable discrete-event simulator for wireless networks by UCLA which provides detailed radio and MAC layers. Table 1 describes the basic parameter settings used in the simulation. The chosen power consumption param correspond to the TR1000 radio from RF Monolithics [20], where the transmission range is set to approximately 20 m. This low-power radio has a data rate of 2.4Kbps and uses OOK modulation [21].

If not mentioned particularly, for most of the experiments, the recall requirement is 0.9. 100 sensors are placed in a terrain of size [200m, 200m] with grid unit 10 m. We observe that in this setting, a node has at most 16 children and the routing tree depth is 6. Every 20 sec, 10% to 50% of nodes fail for a duration randomly chosen from 5 sec to 20 sec. In addition, we model link quality as in [26]: For each directed node pair at a given distance, we associate a link failure probability based on a mean and a variance, assuming that the probability follows a normal distribution. Each simulated packet transmission is filtered out with this probability. Typically, we set the mean of this to be 0.3.

5.2 Experimental Results

The three free parameters in our setting are: changing sensor values, query recall requirements and sensor network conditions; so we evaluate network behavior by varying these factors. Our experiments consist of five parts, testing for: (i) The impact of varying selectivity (varying $|\mathcal{E}|$) (ii) system performance (varying r_q) (iii) system resilience against failure severity (iv) protocol performance as $|\mathcal{S}|$ increases or node density increases, and (v) the impact of re-structuring.

Varying Selectivity.— We study how latency and overhead vary as selectivity changes from 0.1 to 1.0. Figure 7 shows that there is a point (~ 0.7) with minimal latency. If $|\mathcal{E}|$ is low, then the effect of even a few faults is amplified, and repeated rounds are performed to resolve the status (YES/NO) of most nodes; conversely if $|\mathcal{E}|$ is high, then many YES tuples must reach IP; hence, latency is at its worst for these extreme cases. E2E incurs higher latency and consumes more energy than FATE-CSQ uniformly.

Varying Recall Requirement.— In this experiment, we vary r_q from 0.5 to 1.0. Since we are interested in the latency of each protocol in achieving r_q, we set the query period to be 20min, enough for the protocol to meet the requirement—for the set data rate and physical setup.

Figure 8 shows little variation in either latency or energy consumption for LAZY. This is due to its ignorance of query requirements and network conditions. No extra effort is put to achieve higher recall after each sensor reports its value once. The rightmost plot in Figure 8 shows the actual recall provided by each protocol given recall requirement of 0.9. In comparison, E2E checks periodically its current recall and sends feedback to ask for more information if necessary. FATE-CSQ does not issue the STOP signal until the recall requirement is met, which means that retransmission of sensor values or the number of NO continues inside the sensor network. E2E incurs much higher latency and cost than FATE-CSQ, which benefits from its hop-by-hop recovery. We also observe that when the recall requirement is low, E2E consumes less energy than FATE-CSQ; this is because reporting once from each sensor suffices and there is no feedback necessary from the IP. However, in FATE-CSQ, intermediate nodes initiate an additional round before receiving the STOP signal.

Varying Failure Severity.— In this experiment, we investigate how the link failure rate affects performance (Figure 9). We vary the mean of the link failure probability from 0.1 to 0.5. Since LAZY does not try to ensure that the recall requirement is met, its performance provides a comparative baseline. For FATE-CSQ and E2E, both latency and energy consumption increase as the link failure rate increases, since more feedback messages and re-transmissions are required to meet r_q. FATE-CSQ outperforms E2E uniformly.

Node Placement	Grid			
MAC Layer	CSMA			
Radio Layer	RADIO-ACCNoISE			
Propagation Model	TWO-RAY			
Radio Range (m)	20			
Radio Bandwidth (kbps)	2.4			
Radio Power (mW)	xmit.	recv.	idle	sleep
	14.88	12.50	12.36	0.016

Table 1: Simulation settings

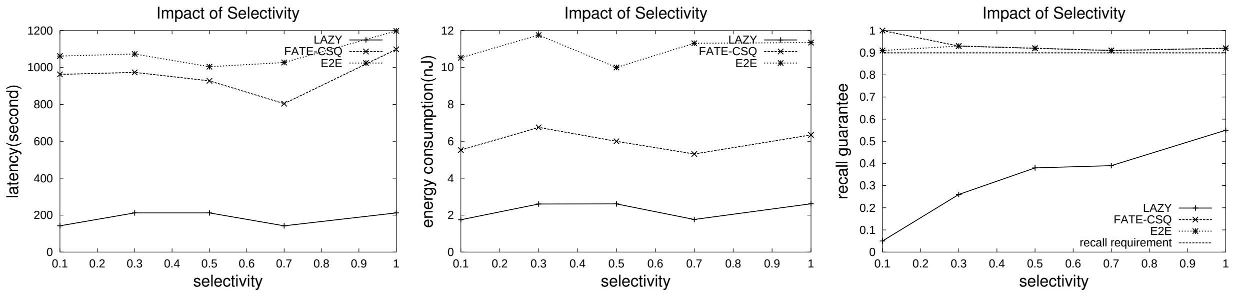

Figure 7: Impact of selectivity

System Scalability.— In this experiment, we vary the grid unit from 5 to 15 m (note that the sensor radio range is set to be 20 m in this simulation). As the grid unit increases, the tree depth varies from 2 to 18 and the maximum number of children varies from 57 to 2. Figure 10 shows that in general both the latency and the energy consumption increase as the node density decreases. However, there exists a point (grid unit of 5 m in this case) where the latency and energy consumption are the lowest. This is because when the grid unit is small, node density is high and one node has more neighbors within its radio range, therefore the height of the routing tree is small. To avoid collisions, the total time needed for these children to transmit their data is long. In contrast, when the grid unit is large, there are more hops between most sensors and the IP, and more time is needed to get most data received by the IP.

We also vary the number of nodes from 25 to 200. As this increases, both latency and energy consumption increase. All three protocols follow a similar trend, but FATE-CSQ outperforms E2E consistently (Figure 11).

Frequency of Re-Structure Phase.— As we have discussed previously, the re-structure phase represents a neccessary overhead in order to maintain the health of the network. In this experiment we vary a parameter β from 0.6 to 1.0: the re-structure phase is initiated if the last period used more than $\beta\tau$ time to meet the r_q requirement. We observe from Figure 12 that there is a point where the energy consumption is lowest. This confirms our intuition in Section 3. We also observe that the energy consumption with β being small (0.6 in this case) is lower than with β being large (0.9 or 1.0). This is because the normalized en-

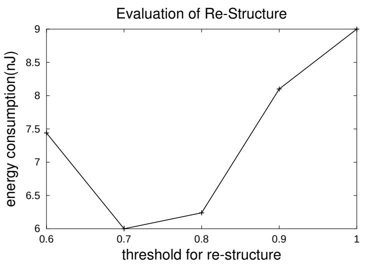

Figure 12: Impact of Re-structure

ergy consumption for topology discovery is about 1.2 uJ in general. To be conservative, it is more efficient to initiate topology re-discovery more frequently if the optimal point is not found.

Performance Summary.— In summary, FATE-CSQ outperforms E2E and LAZY uniformly under varying selectivity, recall requirements and, network conditions, scaling well as the number and density of nodes increase.

6 Related Work

Stann and Heidemann [22] have recognized the importance of faults in sensor networks, and have proposed a solution for a reliable transport protocol that fragments and re-assembles a data object reliably over the sensor network. Their work concerns the transport of an object broken down into chunks, whereas we deal with multiple small objects (data tuples) from many different sensors. Completely reliable data transmission is not necessary in our case, and thus re-transmission can end when the required level of quality is reached. [22] also considered analytically end-to-end and hop-by-hop protocols but under the

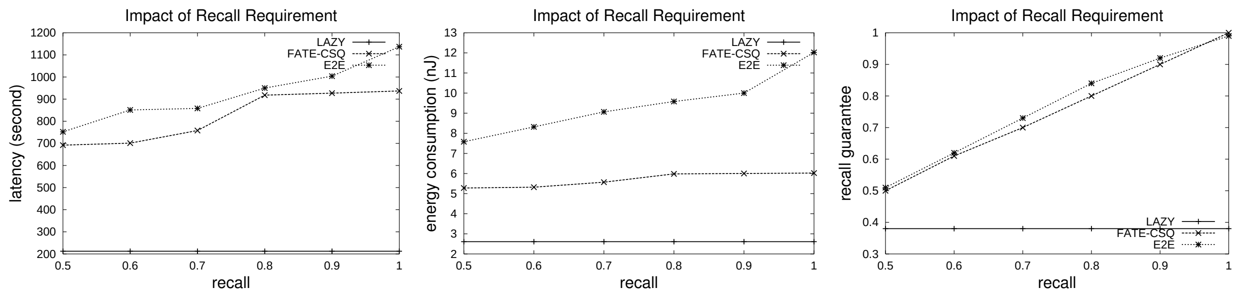

Figure 8: Impact of recall requirements r_q

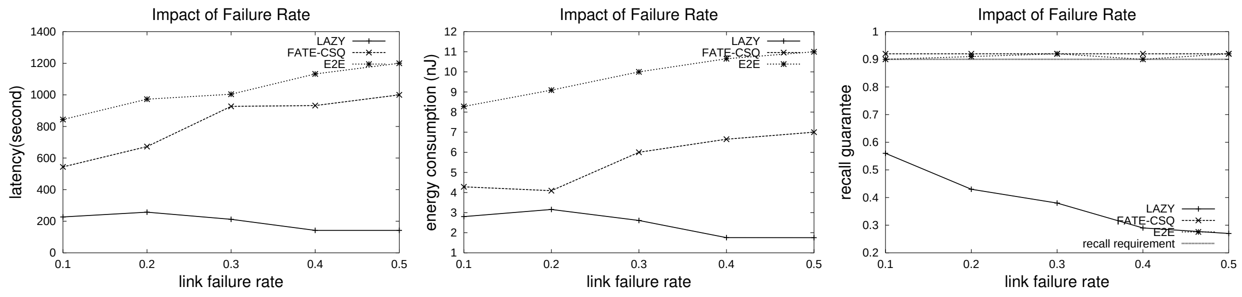

Figure 9: Impact of link failure rate

assumption that an infinite number of retries could be attempted; our analytical treatment of the problem compared three different policies under the additional time constraint imposed by the periodicity of the continuous query. Kim et al. [13] have also considered reliable transfer in sensor networks, proposing the use of both re-transmission and erasure codes which are used to reconstruct a message of M packets if any M from $M + R$ packets are received. In our case, reliability is not needed to reconstruct a long message broken into many pieces, but rather to collect multiple small pieces (tuples) from a number of different sources. Additionally, using alternative routes to send data is not feasible in our case, since this would require additional book-keeping to ensure that duplicate tuples are not counted more than once in computing the recall level achieved by the system.

Limiting the effects of faults has been studied in the past by TAG [16] and SKETCH [6] for aggregate queries. In TAG, a routing tree is formed during query dissemination phase. Later, a sensor node selects a new parent if (1) the quality of the link with his parent is significantly worse than that of another potential parent, or (2) it has not heard from its parent for some period of time. SKETCH uses a DAG instead of a tree for data delivery. Given that most nodes have multiple parents in a DAG, an individual link or node failure has limited effects. A robust technique for computing duplicate sensitive aggregates was proposed by combining multi-path routing and duplicate insensitive sketches.

More recently Bawa et al. [3] have identified the ill-defined semantics of current best-effort algorithms over dynamic networks (including P2P systems and sensor networks) and have sought to formalize these with a correctness criterion called *single-site validity*. [3] deals with node faults (using our terminology). Deshpande et al. [8] have also recently identified shortbacks of traditional best-effort algorithms, with the key observation that not *all* sensor values should be retrieved at all times, to cut down on energy-expensive sensing and communication. Thus, they build a statistical model based on a subset of sensor values which has the additional benefit of being able to predict “missing values.” We believe that this would be useful on top of FATE-CSQ, as it would help estimate the values of the sensors which such a protocol could not recover from the network.

Our work complements *data stream* systems using *load shedding* [23, 1]. A data stream processor often needs to “drop” tuples at the input of operators, if its capacity does not suffice. By taking into account the fraction of tuples dropped in the different branches of the query plan, an attempt is made to recover maximum capacity for a given output stream quality loss. The assumption that the input streams themselves are of perfect quality is not always valid, e.g., for sensor-generated data. In such a case, tuples have been dropped *before* they reach the central monitoring site—not by choice (to recover capacity) but by accident (faults). Our paper gives a bounded estimate of this *a priori* drop rate which can be easily incorporated in e.g., the framework of [23]. Data stream managers can thus become aware of *data quality*, for

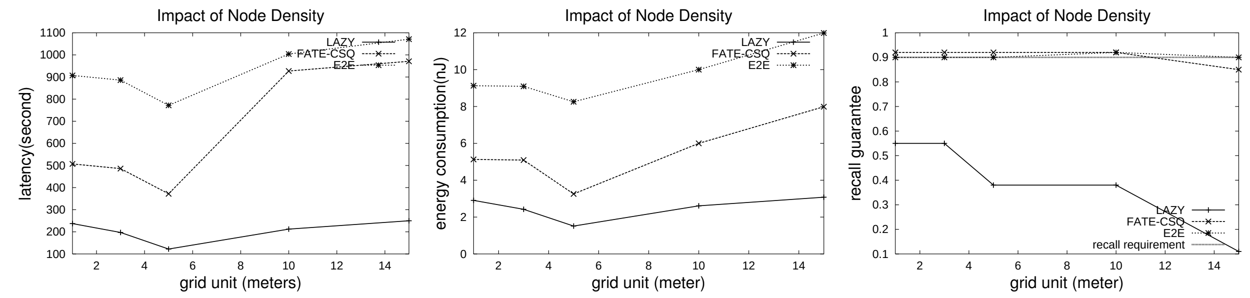

Figure 10: Impact of node density

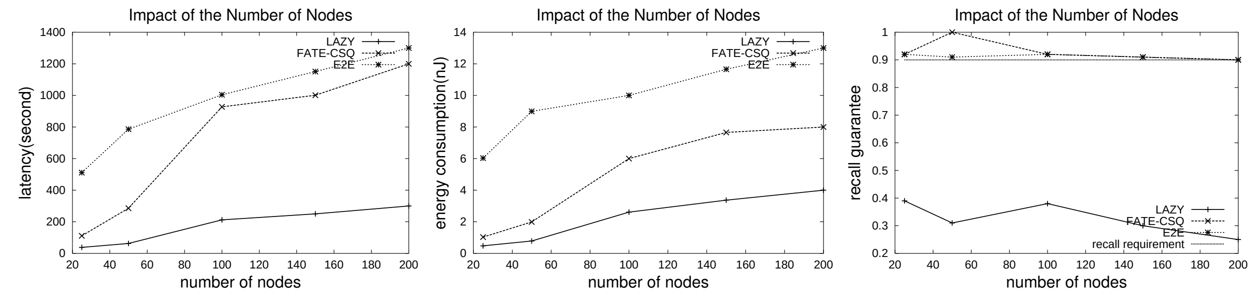

Figure 11: Impact of the number of nodes

instance by choosing not to shed more load from an input stream that is already missing many tuples.

Hop-by-hop error recovery in sensor networks was proposed by PSFQ (Pump Slowly, Fetch Quickly) [25]. Driven by the purpose of controlling, managing or re-tasking sensors, PSFQ aims to provide in-sequence data delivery from the IP to the sensors. Along similar lines, GARUDA [18] also provides IP-to-sensors reliability, unlike our work which aims for sensor-to-IP reliability. In addition, PSFQ assumes that message loss in sensor networks occurs because of poor link quality rather than congestion. However, the urgent need for congestion control has been pointed out while discussing the infrastructure tradeoffs for wireless sensor networks [24]. ESRT (Event-to-Sink Reliable Transport) [27] aims to provide congestion control in sensor networks by adjusting sensor reporting frequency based on current network congestion and application specific reliability requirements. With the same objective, CODA (Congestion Detection and Avoidance) [7] provides an energy efficient congestion control scheme which decouples application reliability from control mechanisms. Our work is more similar to ESRT in that we aim to achieve overall application quality requirements. In our application of CSQs, in-sequence delivery is not needed, hence PSFQ's guarantees are superfluous.

Providing reliable data delivery has also been addressed by routing protocols. Braided Diffusion [10] maintains multiple "braided" paths as backup. When a node on the primary path fails, data can go on an alternate path. GRAB (Gradient Broadcast) [28] ensures robust data delivery through controlled mesh forwarding. It controls the "width" of the mesh, thus the degree of redundancy in forwarding data. Reliable routing does not differentiate data and enforces reliable delivery of each piece of data, which is neither efficient nor necessary. Interesting work has been done to evaluate the impact of link quality estimation and neighborhood table management on reliable routing in sensor networks [26].

Sympathy [19] was developed at UCLA for debugging and detecting failures in sensor networks. It analyzes failures to uncover their causes. Our protocol assumes no a priori knowledge of fault conditions, and attempts to correct faults wherever they occur; this may lead to wasted effort, e.g., for a long-term node fault. Information output by a tool like Sympathy, could be used to react to faults more intelligently.

7 Conclusions

We developed a protocol which provides a quality guarantee expressed as the fraction of the exact answer set of a continuous query returned to the user. We use a hop-by-hop feedback/re-transmission scheme motivated by our analytical study of alternative methods. We evaluated our FATE-CSQ protocol against a simpler one that doesn't consider faults, and a smarter end-to-end protocol that does not use in-network processing to localize and quickly fix the effects of faults. Our emphasis has been on the need for clear semantics of data obtained from the network via queries: total retrieval of relative data may be impossible, but quantifying the accuracy of answers will go some way to making it interpretable to users of the system. Potential extensions of our work include the consideration of more diverse types of queriesand the study of the resilience of different physical layouts and network topologies to faults.

8 Acknowledgments

This work was supported by the National Science Foundation under Award Numbers 0331707 and 0331690.

References

[1] B. Babcock, M. Datar, and R. Motwani. Load shedding for aggregation queries over data streams. In *Proc. of ICDE Conference*, 2004.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley, 1999.

[3] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani. The price of validity in dynamic networks. In *SIGMOD Conference*, 2004.

[4] A. Boulis, S. Ganeriwal, and M. B. Srivastava. Aggregation in sensor networks: an energy - accuracy tradeoff. *Sensor Network Protocols and Applications, Special Issue of Elsevier Ad Hoc Networks Journal*, 2003.

[5] C.-Y. Chong and S. Kumar. Sensor networks: evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8), 2003.

[6] J. Considine, F. Li, G. Kollios, and J. Brers. Approximate aggregation techniques for sensor databases. In *Proc. of IEEE ICDE*, 2004.

[7] C.Y.Wan, S. B. Eisenman, and A. T. Campbell. Coda:congestion detection and avoidance in sensor networks. In *SenSys Conference*, 2003.

[8] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model driven data acquisition in sensor networks. In *VLDB Conference*, 2004.

[9] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *Proc. of SenSys*, 2003.

[10] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *ACM MCCR*, 1(2), October 2002.

[11] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Mobile Computing and Networking*, 2000.

[12] R. Kannan, R. Kalidindi, S. Iyengar, and V.Kumar. Energy and rate based mac protocol for wireless sensor networks. *ACM SIGMOD Record*, 32(4), December 2003.

[13] S. Kim, R. Fonseca, and D. Culler. Reliable transfer in wireless sensor networks. In *The First IEEE International Conference on Sensor and Ad hoc Communications and Networks*, October 2004.

[14] I. Lazaridis and S. Mehrotra. Capturing sensor-generated time series with quality guarantees. In *Proc. of ICDE Conference*, 2003.

[15] I. Lazaridis and S. Mehrotra. Approximate selection queries over imprecise data. In *Proc. of ICDE Conference*, 2004.

[16] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *USENIX OSDI*, 2002.

[17] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD Conference*, 2003.

[18] S. J. Park, R. Vedantham, R. Sivakumar, and I. F. Akyildiz. A scalable approach for reliable downstream data delivery in wireless sensor networks. In *ACM MobiHoc Conference*, 2004.

[19] N. Ramanathan, K. Chang, L. Girod, R. Kapur, E. Kohler, and D. Estrin. Sympathy for the sensor network debugger. In *SenSys Conference*, 2005.

[20] RT Monolithics Inc., http://www.rfm.com/. *ASH Transceiver TR1000 Data Sheet.*

[21] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava. Topology management for sensor networks: Exploiting latency and density. In *ACM MobiHoc*, 2002.

[22] F. Stann and J. Heidemann. Rmst: Reliable data transport in sensor networks. In *Proceedings of the First International Workshop on Sensor Net Protocols and Applications*, pages 102–112, Anchorage, Alaska, USA, April 2003. IEEE.

[23] N. Tatbul, U. Cetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *VLDB Conference*, 2003.

[24] S. Tilak, N. N. Abu-Ghazaleh, and W. Heinzelman. Infrastructure tradeoffs for sensor networks. In *Proc. of WSNA*, 2002.

[25] C. Y. Wan, A. T. Campbell, and L. Krishnamurthy. Psfq: A reliable transport protocol for wireless sensor networks. In *Proc. of ACM WSNA*, 2002.

[26] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys Conference*, 2003.

[27] O. B. A. Y. Sankarasubramaniam and I. F. Akyildiz. Esrt: Event-to-sink reliable transport in wireless sensor networks. In *Proc. of ACM MobiHoc*, 2003.

[28] F. Ye, G. Zhong, S. Lu, and L. Zhang. Gradient broadcast: A robust data delivery protocol for large scale sensor networks. *ACM WINET*, 11, 2003.

[29] X. Zeng, R. Bagrodia, and M. Gerla. Glomosim: a library for parallel simulation of large-scale wireless networks. In *Proc. of PADS*, 1998.

[30] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *SenSys Conference*, 2003.