This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE SECON 2006 proceedings.

RideSharing: Fault Tolerant Aggregation in Sensor Networks Using Corrective Actions

Sameh Gobriel, Sherif Khattab, Daniel Mossé, José Brustoloni and Rami Melhem
Computer Science Department, University of Pittsburgh
{*sameh, skhattab, mosse, jcb, melhem*}*@cs.pitt.edu*

***Abstract*— *In Wireless Sensor Networks (WSNs), the users' objective is to extract useful global information by collecting individual sensor readings. Conventionally, this is done using in-network aggregation on a spanning tree from sensors to data sink. However, the spanning tree structure is not robust against communication errors; when a packet is lost, so is a complete subtree of values. Multipath routing can mask some of these errors, but on the other hand, may aggregate individual sensor values multiple times. This may produce erroneous results when dealing with duplicate-sensitive aggregates, such as SUM, COUNT, and AVERAGE.*

In this paper, we present and analyze two new fault tolerant schemes for duplicate-sensitive aggregation in WSNs: (1) Cascaded RideSharing *and (2)* Diffused RideSharing. *These schemes use the available path redundancy in the WSN to deliver a correct aggregate result to the data sink. Compared to state-of-the-art, our schemes deliver results with lower root mean square (RMS) error and consume much less energy and bandwidth. RideSharing can consume as much as 50% less resources than hash-based schemes, such as SKETCHES and Synopsis Diffusion, while achieving lower RMS for reasonable link error rates.*

I. Introduction

The convergence of sensing and wireless technologies has enabled the powerful networking paradigm of Wireless Sensor Networks (WSNs). WSNs are expected to have significant impact on the efficiency of many military and civilian applications, such as combat field surveillance, environmental monitoring, security and disaster management, data gathering, and alarm systems [2]. In large-scale WSN deployments, sensor measurements are often aggregated within the network (in-network processing) to filter redundancy and reduce communication overhead and energy consumption [15], [16], [22].

However, communication errors are frequent in WSN [28], and when a spanning-tree is used for aggregation (e.g., [18]), a packet loss can result in the loss of the result of a complete subtree. Multipath routing can overcome losses by duplicating and forwarding each sensor measurement over multiple paths [12]. Some aggregate functions, such as MIN and MAX, are unaffected by duplicates, but some others, such as SUM, COUNT, and AVG (short for AVERAGE), are duplicate-sensitive and may produce wrong results with duplicate aggregation. To handle duplicate-sensitive aggregation, the hash-based framework has been proposed, most notably SKETCHES [6] and Synopsis Diffusion [19], [21].

In this paper, we propose the RideSharing (RS) scheme for fault-tolerant, duplicate-sensitive aggregation in WSNs. RS uses the inherent redundancy of the wireless medium; when a packet is lost between two sensors because of a link error, it is possible that one or more other sensors have correctly overheard the packet. If some of them are yet to send their own values, they correct the error by aggregating the missing value into theirs. As a result, error recovery has no overhead because the lost packet is aggregated (or RideShares) with another packet to be transmitted.

We present two distributed mechanisms to support duplicate-sensitive aggregation by ensuring that each sensor value is aggregated into the final result at most *once* (i.e., no more than 100% of the value is aggregated). In *cascaded RideSharing*, at most one overhearing sensor corrects an error, while in *diffused RideSharing* each overhearing sensor aggregates a share of the missing value, whereby the sum of these shares never exceeds the value to be corrected. It should be mentioned that, although the RS schemes are designed to handle duplicate-sensitive aggregations (e.g., SUM, COUNT, AVG), it can similarly handle the non-problematic case of duplicate-insensitive aggregation (e.g. MAX, MIN) where the packet can be aggregated in *all* the neighbors' messages.

We compare RS with state-of-the-art schemes for duplicate-sensitive aggregation in WSNs, namely SKETCHES [6] and Synopsis Diffusion [19], [21]. Through simulations, we show that RS can consume as much as 50% less resources (energy and bandwidth) than do hash-based schemes while delivering more accurate results for reasonable link error rates (up to 20%). Moreover, RS significantly outperforms hash-based schemes for all link error rates when only a few sensors' values are requested. In fact, this is important because having a subset of nodes participate in the query reply is a frequent case and can happen, for example, when a query is dispatched to the WSN and not all sensors satisfy the particular request (e.g., the sensor values do not match the SQL WHERE clause).

The rest of the paper is organized as follows. In the next section, we review reliable aggregation schemes proposed in the literature. Section III presents our RS schemes. Section IV describes our simulation study and Section V concludes the paper.

II. Reliable Aggregation in WSNs

In this section, we review related work addressing the problem of reliable aggregation in WSNs. Typically, a "data sink" floods the WSN with a query, such as MAX, SUM, or AVG, and sensors provide the data sink with the information they gather either periodically or per-query. Each sensor could send its $\langle id, value\rangle$ pair to the data sink, but this would increase the data being transmitted in the network. In-network aggregation can reduce this communication overhead substantially [14], [18]. Sensors are arranged in a spanning tree rooted at the sink, each intermediate node receives messages from its children, computes an aggregate (e.g., SUM) of its children's and own values, and then forwards to its parent only a single message with the aggregate value. Although efficient, the aggregation spanning trees are not robust against communication errors, which are very common in sensor networks. When a packet is lost because of a link error, a complete subtree of values is lost, possibly leading to an incorrect aggregate result [6], [18], [19], [21].

Several mechanisms have been proposed for reliable data delivery in WSNs. For example, error-correction [7], [20], [22], [23], retransmission [17], [25], [27], multipath routing [4], [6], [21], partial parents [18], and hash-based [6], [19], [21] schemes.

Forward Error Correction (FEC) increases communication reliability in WSNs by efficiently reducing the effective link error rate [22]. Therefore, it benefits any reliable aggregation scheme including ours. Packet-Combining [7] is an error correction scheme which, similar to our RS, uses the inherent redundancy of the wireless medium. Nodes buffer all received packets even if corrupted; the original packet may then be recovered from combining two or more corrupted versions. However, Packet-Combining across multi-hops cannot handle data aggregation because it assumes packet contents are not altered by intermediate nodes. On the other hand, application-level error correction [20], [23], exploits spatio-temporal correlations of sensor reports to predict missing values. Our RS scheme improves data accuracy even when prediction models are hard to develop or when unpredicted events occur.

Examples of fault-tolerant protocols that use retransmission are PSFQ [27], RMST [25], and SPMS [17]. PSFQ and RMST use retransmission for hop-by-hop error recovery, while SPMS recovers from errors by retransmission over backup routes. Retransmission, can suffer from two drawbacks: (1) delayed query response, because each level in the routing tree waits for retransmissions before proceeding with its own transmission; and (2) packet overhead, because some kind of handshake (e.g., ACK packets) is used for error detection [18].

In ring-based multipath routing [6], [21], sensors are arranged into a ring topology, where a node located in ring i attaches itself to some nodes (parents) in ring $i-1$. When a node has something to transmit, it multicasts to all its parents, and each parent aggregates the received data with that received from other children before forwarding the aggregate result. Multipath routing, however, produces wrong results for *duplicate-sensitive* aggregates (e.g., COUNT, SUM, AVG); because a node value is aggregated over each path, it can be incorporated multiple times into the final result.

Exor [4] is a fault-tolerant routing protocol for adhoc networks. It assumes that when a packet is lost some other nodes may have correctly overheard it and will then try to re-route the lost packet. However, Exor requires the destination to detect and drop duplicates and thus, it can not be used with data aggregation.

Partial parents, an optimization of the TAG scheme [18], uses the ring topology as well. Each data value is decomposed into a number of fractions, and each fraction is aggregated by a distinct parent. Results in [6] show a small improvement of this optimization over TAG; although each link error results in losing a smaller value compared to the spanning-tree, more links are used and thus more errors occur.

Hash-based schemes [6], [19], [21] are the most relevant related work to RS. These schemes have been developed to handle the same problem of fault-tolerant, duplicate-sensitive aggregation in sensor networks. In this paper we compare RS to a hash-based scheme, namely Synopsis Diffusion (SD) [21].

The main idea of hash-based schemes is to transform duplicate-sensitive aggregation functions into duplicate-insensitive ones. For instance, the COUNT function, which is duplicate sensitive, is transformed into a bitwise *OR* operation (*OR*ing is duplicate-insensitive: $x\ OR\ y = x\ OR\ (y\ OR\ y\ OR\ y\ \cdots)$). The *OR*ed bit vectors are generated by hashing each sensor *id* into *one* bit in a hash table. Suppose that a number of distinct values have been hashed into the hash table. Each value sets a bit in the hash table to 1. Suppose the position of the least significant 0 in the resulting hash table is z. Then, the number of distinct values is estimated as $\frac{2^z}{0.77351}$ (see [10] for more details). The position of the least significant 0 is a more accurate estimator than the number of 1-bits [10]. To improve the estimation accuracy, a number, m, of hash tables are used, and the values to be counted are distributed among these hash tables. Assume the position of the least significant 0 is z_i in hash table i, and the average of the z_i's is $\hat{z}$. The estimated value in this case is

$$\frac{2^{\hat{z}}}{0.77351} \times m \tag{1}$$

In other words, each of the m hash tables estimates $\frac{1}{m}th$ of the count.

Although they are a great improvement on aggregation in WSNs, hash-based schemes suffer from the following drawbacks: (1) their computation and storage requirements are high; (2) a distinct algorithm has to be devised for each aggregation function; for example, the algorithm used for COUNT can not be readily used for SUM although both functions are simply additions; (3) a relatively large bit-vector (e.g., $O(log(n))$ for COUNT, where n is the total number of sensors) is attached to *each* data message, consuming extra energy and bandwidth; (4) the probability that the correct aggregate result is delivered severely *degrades* when the number of nodes participating in the query decreases (we elaborate on this issue in Section IV).

III. RideSharing Fault-tolerant Aggregation

Our new RideSharing (RS) scheme addresses fault-tolerant, in-network aggregation of duplicate-sensitive functions. RS exploits the inherent redundancy of the shared wireless medium to detect and correct communication errors with low overhead. When a message between two nodes is lost we assume that one or more other sensors have correctly overheard the lost message. When some of the overhearing sensors have not yet transmitted their own values, they can aggregate the missing value into theirs (RideSharing).

It should be mentioned that the assumption of overhearing sensors (which has also been adopted in other work [4], [7], [12]) does not constrain RS to only dense networks because such assumption is easily justified when a sensor has more than one neighbor within its range. In Section III-F we present an optimization to be used in low-density WSNs, and in Section IV we simulate RS for different network densities.

It should also be noted that, RS assumes, similar to the related work mentioned in Section II, that the network is static on a per query basis. In other words, the state of the network is assumed to be unchanged for a single query, otherwise, the query aggregate result will be "stale" data.

A. Track Topology

RS organizes sensors in a *track* graph [9]. As shown in Figure 1, the data sink is in track 0, sensors one hop away from the data sink are in track 1, and so on. A directed edge from a child C to a parent P_1 indicates that P_1 and C are within each other's radio range and that P_1 listens to C's communication. The difference between our track topology and the ring topology [6], [21] is that, in addition to edges between adjacent levels, tracks also have edges between sensors of the same track.

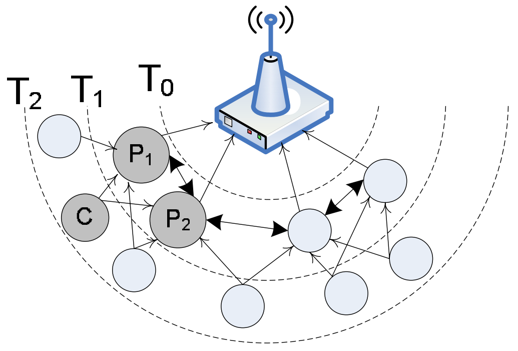

Fig. 1. Track Topology

Serving different purposes, edges are classified into three types: *primary*, *backup*, and *side* edges; primary and backup edges are between adjacent tracks (between a sensor node and its parents). Side edges are within the same track (among parents). Each sensor selects one parent (and correspondingly one edge) as its primary parent and zero or more parents as backups. Primary edges form a spanning tree and are used as long as no communication error occurs. If an error occurs in a primary edge, it is possible that some backup edges have successfully delivered the sent value. Parents coordinate using side edges so that the missing value is aggregated at most *once* (i.e., no more than 100% of the value is aggregated). It should be noted that a sensor can be a primary parent for some children and at the same time a backup parent for some others. Also, we assume that errors occur independently in primary, backup and/or side edges.

B. Error Detection and Correction

Errors are detected using a small *bit vector* that each parent attaches to each data message it sends. The bit vector efficiently encodes the *ids* of children whose values have been correctly received and aggregated. By overhearing the bit vectors over side edges backup parents detect link errors when one or more children are missing from the bit vector. Each parent determines the bit positions of its children inside the other parents' bit vectors during topology construction, whereby each parent broadcasts children *ids* and their bit positions inside its bit vector.

The bit vector contains two bits for each child of which the sensor is primary or backup parent. The first bit, the *e-bit*, indicates error in the child's primary edge. If the primary parent does not receive from a child, it sets the e-bit to 1. Overhearing the primary parent signal an error, a backup parent sets its e-bit to 1 as well to propagate the error signal. The other bit, the *r-bit*, indicates that the sensor is correcting or helping correct the error. The detailed use of the e-bit and the r-bit will be explained in the next subsections.

We note that there is a difference in functionality between what we call a *backup-parent* and what a backup means in fault-tolerance literature. Traditionally, a backup is a stand-by element which operates only whenever an error is detected. In our case, the backup-parent is an active sensor having its own children and aggregating its value with the values of its children whether or not an error occurs; when an error is detected, the backup accounts for the missed value in the message that it is going to transmit anyway, that is, without sending extra messages. As a result, the RS overhead is only associated with error detection (receiving children's and parents' messages), while there is no overhead associated with error correction.

C. Illustrating Example

As an example, Figure 1 shows a sensor C in track T_2 with two parents, primary P_1 and backup P_2. Assuming no error, both P_1 and P_2 receive C's value, but only P_1 aggregates it. Now, assume a link error in the primary edge. P_2 will receive the bit vector of P_1 over the side edge $P_1 \leftrightarrow P_2$, detect that C is missing, and correct the error by aggregating C's value into its own.

To support duplicate-sensitive aggregation, we ensure that errors are corrected in such a way that every sensor reading is aggregated at most *once* (again, this means that no more than 100% of the value is aggregated). When there is only one backup parent, as in the previous example, the solution is trivial. On the other hand, handling this issue for more than one backup parent requires coordination between the parents. We propose two mechanisms, namely cascaded RS and diffused RS, to achieve such coordination. These two schemes are explained in details in the next two sections.

D. Cascaded RideSharing

In cascaded RS, as long as no error occurs, primary parents aggregate and forward their children's values. When an error occurs (which can be done by checking the e-bit and r-bit) in a primary link, each backup parent decides whether or not to correct the error based on its order in a *correction sequence*. This correction sequence can be, for example, an ascending order of parent ids. The first backup parent in this sequence (parent with smallest id) attempts error correction first. If the first backup parent does not receive the child's value or does not detected the error, the second backup parent attempts the correction, and so on. Deciding the correction sequence based on other criteria (e.g., link qualities) and handling parent joins and leaves is beyond the scope of this paper.

The cascaded RS can be viewed as if each backup parent is assigned a *virtual token* (for each child). The token is released when the parent transmits without aggregating the child's value. Every parent that hears a token released *acquires* the token. A backup parent aggregates a child's value and sets the r-bit if (**1**) it has received a bit vector with the e-bit set (indicating an error in the primary link), (**2**) it has correctly received the child's value, *and* (**3**) it has acquired tokens of all parents preceding it in the correction sequence. Token release is detected when the r-bit is unset. To avoid multiple counting, if a parent fails to acquire preceding parents' tokens, it takes no corrective action.

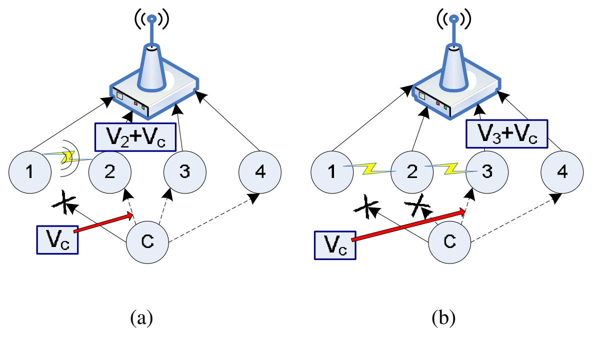

Fig. 2. Cascaded Ridesharing

Figure 2 depicts an example. Child C has a primary parent, 1, and three backup parents, 2, 3, and 4. The correction sequence is 2, 3, 4. As long as no error occurs in the primary link, C's value, V_c, is aggregated at its primary parent. In Figure 2(a), an error in the primary link triggers the primary parent to signal the error by setting the e-bit to one upon transmitting its own value (and the value of its other children, if any). Receiving the primary parent's bit vector, parent 2, detects the error and corrects it by aggregating V_c. Parent 2 sets both r-bit and e-bit to 1 and sends the aggregated value with the piggybacked bit vector. Although parent 3 detects the error, it refrains from correcting it after it hears the bit vector of parent 2 with both bits set.

Figure 2(b) depicts two link errors from C to parents 1 and 2. In this case, parent 2 fails to correct the error and releases its token by setting r-bit to zero. When parent 3 receives parent 2's bit vector, it detects both the error and the failure of node 2 to correct. Acquiring node 2's token, node 3 aggregates V_c and sets both r-bit and e-bit to one in order not to release its token.

In the previous example, two desirable properties hold: (1) each parent's sending order is the same as its correction order and (2) all parents are within range. In Section III-F we propose optimizations to guarantee these properties. However, below we argue by contradiction for the correctness of RS, even if these properties do not hold.

Assume that more than one parent aggregates the child value (i.e., corrects the error in transmission). Consider any two of them x and y and assume without loss of generality that x sends before y. If x is before y in the correction sequence and x corrects the error, y will not acquire x's token because x will not release it. Thus, y will not correct contradicting the initial assumption. On the other hand, if y precedes x in the correction sequence, x will not acquire y's token because x sends first. Hence, x will not correct, similarly contradicting the assumption. The above argument is valid even with errors between parents or if some parents are outside each other's range.

E. Diffused RideSharing

Another approach to ensure that corrective actions avoid duplicate aggregation is to *divide* the child's value to be corrected among backup parents. For example, if the aggregation function is SUM, *each* backup parent aggregates a *part* so that the sum of the aggregated parts equals the child's value. If a backup parent does not detect the error or has not received the child's value, it will not aggregate its share, while the remaining parents adjust their shares to compensate for the missing part.

Each backup parent is assigned a *virtual share* of the child's value to be corrected, so that the child's value is divided (e.g., equally) among its backup parents. So, for instance, if the child's value is V_c and there are 3 backup parents, each virtual share is $virtual\_share = \frac{V_c}{\#backup\_parents} = \frac{V_c}{3}$. A backup parent its virtual share only when it has detected an error *and* has correctly received V_c; it then sets both e-bit and r-bit to one. When a parent does not correct the value, its virtual share is further divided among other parents who have not yet transmitted. To implement this compensation, each backup parent maintains a counter of the *remaining parents*, which are backup parents, including itself, that it has not yet heard; when it hears a parent with either r-bit or e-bit set to zero (indicating that the parent has not aggregated its virtual share), it increases its virtual share by $\frac{virtual\_share}{\#rem\_parents}$; because it is a distributed algorithm, a parent does not know exactly what is the current share of another parent, so it uses its current virtual share as an estimator of the missing parent's share.

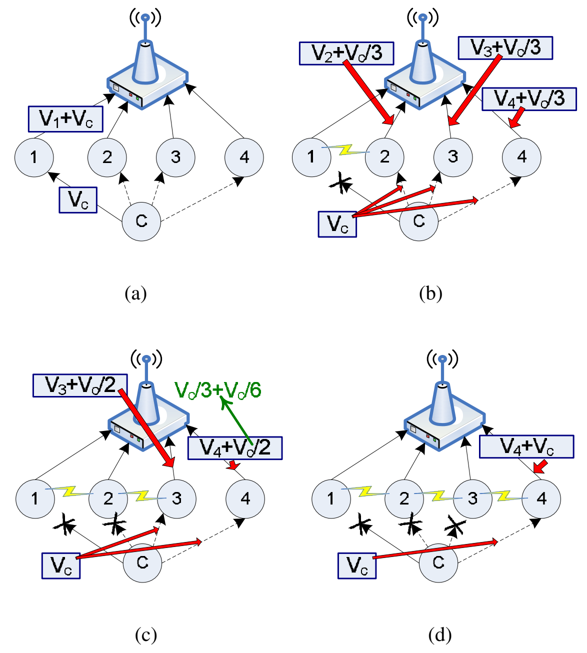

Fig. 3. Diffused RideSharing

For example, in Figure 3(b), when an error occurs between child C and its primary parent, V_c is divided among the three backup parents, so that each aggregates one third of C's value. If parent 2 has not received V_c (Figure 3(c)), it sets the r-bit to zero. Each of parents 3 and 4 adjusts its virtual share to one half $(= \frac{1}{3} + \frac{1}{3\times2})$ upon hearing 2's bit vector. If both links from C to 2 and 3 are in error, parent 4 adjusts its part and aggregates the whole V_c as shown in Figure 3(d).

As a proof of correctness, the total aggregated value of each child never overshoots V_c. If there is no error, V_c is aggregated only at the primary parent. Upon detecting an error, if all backup parents aggregate their shares, the total aggregated value is also V_c, the sum of the virtual shares. When a parent fails to aggregate its share, the virtual shares of the remaining backup parents are increased. The total increase, however, never exceeds the missed share, because the local remaining parents counter (the denominator) never underestimates the actual value.

F. RideSharing Enhancements

In RS, backup parents correct primary link error. RS's fault-tolerance can be further improved by applying any of the following optimizations:

1) **Co-tracking:** Normally, each child selects its parents from the adjacent track that is closer to the data sink (e.g., in Figure 1, C in track T_2 selects P_1 and P_2 in track T_1). When there is only one reachable sensor in that track, the child selects parents from its own track. In such situations, *co-tracking* allows the child's value to be corrected if there is an error in the primary link. The primary parent is selected from the same track, so that the backup parent in the adjacent track will have a chance to correct an error in the primary link. To avoid loops, whereby two sensors end up selecting each other as primary, we ensure that the parents selected from the same track have a higher node *id* than the child.
2) **Parent Clique:** The ability of a backup parent to correct an error depends on hearing the other parents. Thus, to increase the probability of error correction, it is possible for each child to select parents that hear each other (i.e., the parents thus form a *clique*). We add this optimization because a backup parent needs to overhear a bit vector with the e-bit set before attempting to correct an error. In cascaded RS, it has to acquire the preceding parents' tokens and in diffused RS, by hearing other parents, each backup parent maintains an accurate estimate of the number of remaining parents.
3) **Transmission Order:** To increase the probability that an error is corrected, backup parents should send their messages after primary parents. Such transmission order can be achieved deterministically by a TDMA scheme or probabilistically by a prioritized contention-based scheme.
   A naive but inefficient (longer delay) TDMA allocates a time slot for each node. For example, the schedule starts by the farthest track from the data sink and proceeds inward, whereby the first sensor to send in each track is the one with the smallest *id within* the track followed by the second smallest and so on. In such transmission order, each child selects the primary parent as the parent with the smallest id. A more efficient TDMA scheme (e.g., [3], [11], [26]) can also be used, for example by using graph-coloring algorithm is used to assign transmission slots for nodes. Nodes outside each other range can transmit simultaneously (have the same color). Thus, the schedule is compacted and the end-to-end delay is minimized. In this scheme, the child selects the parents with distinct colors and the primary parent is selected to be the one transmitting first.
   The transmission order can also be enforced with a contention based scheme (e.g., [13]) in which each node is assigned a priority (contention window size) to access the medium. The transmission priorities can be based on node IDs, residual energies or any other criteria. In this case the child picks as primary the parent with the highest priority in the child's parents list.

IV. Evaluation

In our evaluation we compared our RS scheme (cascaded and diffused) with a hash-based scheme, namely Synopsis Diffusion (SD). We present results for two SD schemes, SD-20 [21] and SD-40 [19], the first uses 20 hash tables per message while the second uses 40 hash tables (increased accuracy but with higher overhead). The spanning tree approach [18] (which provides no fault-tolerance but has the least overhead) is used as the basis for the comparison. The following metrics are used to evaluate the performance of the different schemes:

- *Average relative RMS*, that is, the average root mean square error in the aggregate result, normalized to the correct result value. This metric is a good measure of the accuracy of the estimated value.
- *Average energy per node per epoch*, that is, the total transmission, listening, and reception energy consumed per node averaged over the number of epochs. This metric reflects the overhead of each scheme.

We implemented our RS schemes and the SD scheme in CSIM [1] using the TAG simulator [18]. In the TAG model [18], sensor readings are aggregated and sent to the data sink every *epoch*. We present results for COUNT query. It should be mentioned that we are comparing against the best aggregate result for SD, namely COUNT; other aggregates (e.g., SUM) have a larger RMS in SD.

A. Simulation Setup

In our simulation analysis, a number of sensor nodes are randomly distributed in a $300 \times 300\,ft$ area. The radio range of each node is assumed to be $30\,ft$. The data sink is the nearest node to the center. Each simulation run has 100 epochs, 300 *msec* each, and the results shown are the average of 10 runs. Based on the Mica2 motes

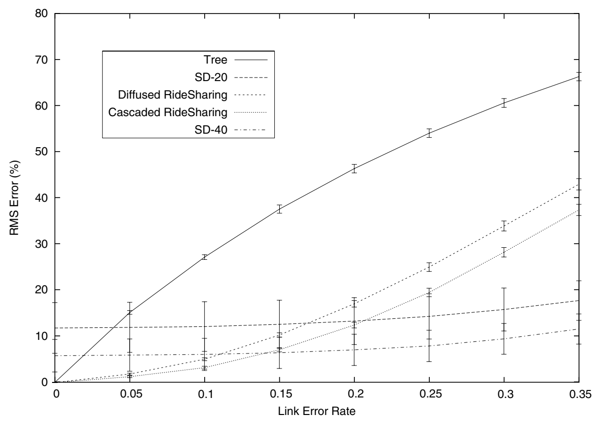

Fig. 4. RMS for 100 % Participation *(parents/node = 3, total nodes = 1000)*

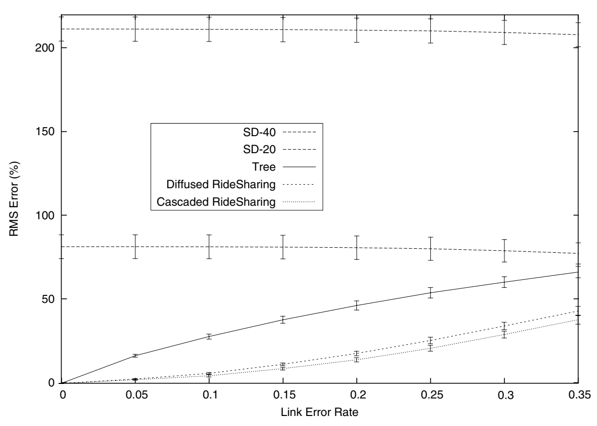

Fig. 5. RMS for 2% Participation *(parents/node = 3, total nodes = 1000)*

power model [24], the power consumption is 65 *mW* for transmission, 21.0 *mW* for listening and reception, and 0 *mW* in sleep mode. The network bandwidth is assumed to be 38.4 *Kbps* [5]. The message size is different based on the scheme used; for the spanning tree it is 2 *bytes*, for RS it is 2 *bytes* + 2 *bits* $\times$ *number of children*, for the SD-20 it is 12 *bytes* (20 *bit vectors* $\times 2$ *bytes each* $\times 0.3$ *compression ratio*) [6], and for SD-40 it is 24 *bytes*. We use the same assumptions for link errors as that used in the SD evaluation [21]: link errors are independent and uncorrelated and links between track 1 and the data sink are error-free. In our simulations we varied the link error rate, the total number of sensor nodes, the number of participating nodes in the query, and the maximum number of parents per node.

B. Experimental Results

1) Accuracy Comparison: First, let's consider the case when the total number of nodes is 1000 and all of them are participating in the query. Figure 4 shows the relative RMS versus the link error rate. The error bars represent 90% confidence intervals. As expected, when the error rate increases the performance of the spanning tree severely degrades. The RMS reaches 67% for a link error rate of 35%. This is because the spanning tree topology is not robust against errors, and if a packet is lost, so is a complete subtree of values, increasing the error in the final result.

SD, on the other hand, is robust to link losses. The value of a node will be delivered as long as there exist at least one error-free path from this node to the sink. SD-20 achieves a relative RMS of about 12.5% at a link error rate upto 10%, while SD-40 achieves 7.5%. At a link error rate of 35%, the relative RMS increases to only 18% for SD-20 and to only 12% for SD-40. (These results match those in [19], [21].) However, it should be noted that, there is always an error in SD even when the network is error-free. This error is associated with the hash operation and is independent from the network.

On the other hand, RS does not suffer from this drawback and achieves a better RMS than SD-20 for link error rates up to 20% (*practical* wireless networks have much lower link error rates [8]). Cascaded RS achieves better relative RMS than diffused RS. This is because some link errors are masked by cascaded RS, while hurting diffused RS. For instance, consider an error between the child and the *last* backup parent to send. This error is masked in cascaded RS if another backup parent has corrected the value before the last backup parent. On the other hand, in diffused RS the virtual share of the last parent will be lost, as all other backup parents have already sent their values.

Next, we consider the experiment when only 20 nodes (out of 1000) are participating in the query (Figure 5). RS provides a stable performance and the RMS in the aggregate result for this experiment is close to that reported in Figure 4. On the other hand, SD has a very serious drawback. The relative RMS of SD-20 jumps to 80% while for SD-40 it is more than 200%. This huge error is much worse than that achieved when using a non-fault tolerant spanning tree. This surprising result can be justified as follows.

In SD-20, the *minimum* count that can be estimated is when hash tables are all zeros. With 20 hash tables, substituting in Equation 1 yields a minimum value of $\frac{20}{0.77351} = 25.8$. When 20 nodes hash their values, each hash table will end up with one bit set to one, assuming perfect distribution of the 20 nodes over the 20 hash tables. If the 1-bit is in the least significant position of each hash table, the estimated count is $\frac{2^1}{0.77351} \times 20 = 51$, or about 130% relative error. It may happen, however, that a 0-bit is in the least significant position of some hash tables, and thus, the position of the least significant

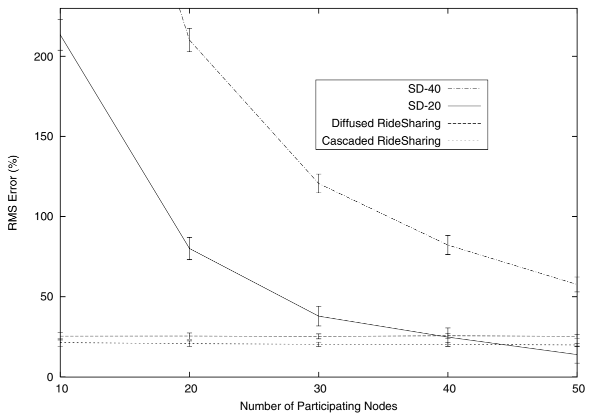

Fig. 6. RMS vs. No. of Participating Nodes *(parents/node = 3, total nodes = 1000, link error rate = 0.25)*

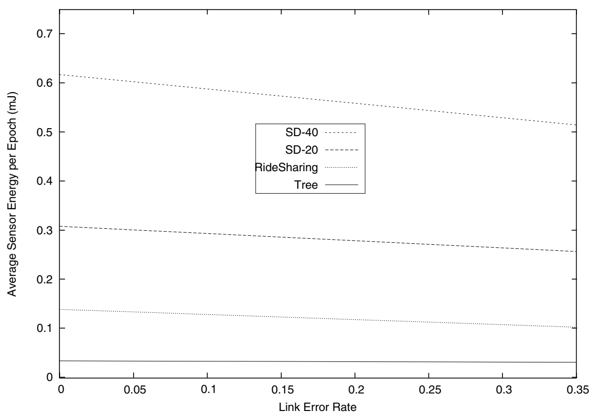

Fig. 7. Average Energy Consumption per Sensor *(parents/node = 3, total nodes = 1000, participation = 100%)*

zero is 0, resulting in lower estimation errors (the 80% error in this experiment resulted from an average least-significant 0-bit position of 0.5 yielding an estimate of $\frac{2^{0.5}}{0.77351} \times 20 = 36$). With similar reasoning, we can show why the RMS of SD-40 is approximately 200%.

To highlight this problem we fixed the link error rate at 0.25 and changed the number of participating nodes. As shown in Figure 6, when the number of participating nodes is low the SD scheme delivers an unacceptable error to the data sink. There is a tradeoff in the design of SD, the number of hash tables has to be large enough to improve the accuracy for queries with a high number of participating nodes. Nevertheless, when a query selects only a small number of nodes a large number of hash tables causes a huge error. It should be mentioned that, determining the number of participating nodes beforehand is not a trivial problem because (1) it is data dependent; for example, a node might send its value only when a significant change occurs [23], and (2) it is query dependent; for example, a query can be dispatched and not all sensors satisfy its WHERE clause. RS, on the other hand, does not need such dynamic tuning and provides an acceptable RMS for any number of participating nodes.

2) Overhead Comparison: In this experiment we evaluate the energy consumed in communication for each scheme. Figure 7 shows the average energy consumption per sensor per epoch in the network, when 1000 sensors participate in the query.

As expected and as shown in the figure, the spanning tree consumes the least energy because each node is transmitting to only one parent, and it has the smallest message size (no added bit vectors). RS consumes approximately 50% of the energy consumed in SD-20 and barely 25% of that consumed in SD-40. Although RS uses the same number of parents as SD, it only adds a small bit vector (2 bits per child). SD, on the other hand, consumes the highest energy overhead, as it uses a large number of hash bit vectors and the resulting message size is large even when compression/decompression is applied at each node.

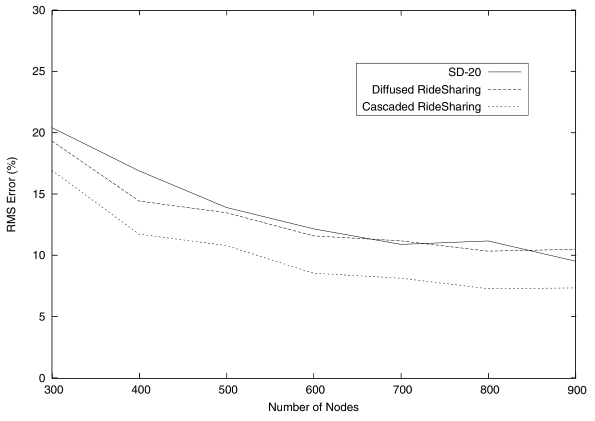

Fig. 8. Effect of Density *(parents/node = 3, participation = 100%, link error rate = 0.25)*

3) Effect of Network Density and Number of parents: First, we consider the effect of the network density on the performance of RS. As shown in Figure 8, we change the total number of nodes deployed in the network. At low density (total nodes = 300) the probability that a node finds 3 parents decrease, and consequently, so does the probability of error correction (because there are no backup parents to mask link errors), as a result, the RMS increases. As the network density increases the accuracy of the RS scheme improves because more nodes can find backup parents. It should be noted that, the effect of the network density on RS is the almost the same as that on SD. This is because any fault-tolerant scheme needs some form of redundancy to be able to detect/correct errors and similarly, SD needs multiple parents per node

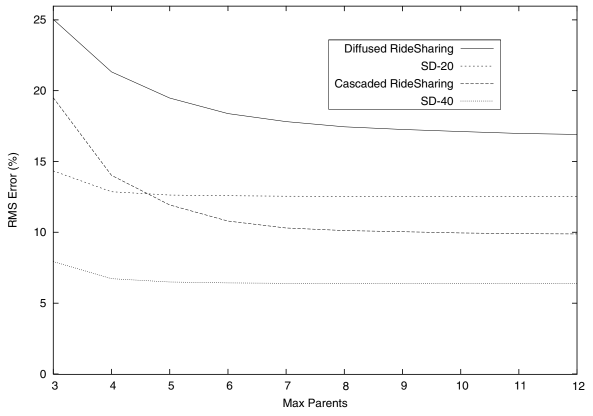

Fig. 9. Effect of number of parents *(total nodes = 1000, participation = 100%, link error rate = 0.25)*

so that the hashed value of a sensor node is not lost when an error occur.

We also evaluate the effect of changing the total number of parents on the performance of RS. As expected and as shown in Figure 9 with a high link error rate of 25%, as the number of parents per node increases the accuracy of both RS and SD improves. However, it is worth mentioning that RS benefits more than SD from increasing the number of parents (RMS decreases from 19.56% @ 3 parents to 11.98% @ 5 parents). This is because increasing the number of parents in SD decreases the error associated with the network, but on the other hand, it has no effect on the error associated with the hash operation and the RMS curve for SD flattens as this is the maximum accuracy that can be delivered using this scheme.

4) Optimizations Effect: The results presented up to this point are for RideSharing with all the three enhancements described in Section III-F. Next, we show the effect of each individual optimization on the performance of our proposed RS schemes. The link error rate is set to 15% for this experiment, the total number of nodes is set to 1000 and all of them are participating in the query.

TABLE I
EFFECT OF OPTIMIZATIONS ON THE RELATIVE RMS

Optimization	Diffused RS	Cascaded RS
None	24.7976	26.5212
Ordering	13.7535	11.3264
Parent Clique	24.178	25.9831
Co-tracking	27.2981	29.0243
All	10.3167	7.14078

Table I shows the relative RMS of both cascaded and diffused RS with no optimizations, when each optimization is applied individually, and when all three optimizations are applied. The transmission ordering optimization has the highest improvement among the others. The co-tracking optimization, by itself, has a negative impact on the relative RMS. In co-tracking, a child selects a sensor (with a higher *id* than itself) from its own track as primary parent. This selection enhances the probability of error correction when the selected primary parent sends *after* the child. Without the transmission order, however, the primary parent may send *before* the child, resulting in losing the child's value because it is never aggregated. The probability that the co-tracked primary parent sends before or after the co-tracking child is the same. Hence, the benefit of co-tracking and the loss of the child's value cancel each other. Moreover, the co-tracking increases the number of hops the message travels and, hence, increases the probability of error. As a result, co-tracking without the ordering optimization delivers a higher RMS.

We note that when no optimization is applied, diffused RS performs slightly better than cascaded RS in this setting. This is because diffused RS is more robust to arbitrary transmission order. This can also be shown from the higher benefit of the ordering optimization in the case of cascaded RS (from 24.79% to 13.75%) compared to diffused RS (from 26.52% to 11.32%).

V. Conclusion

In this paper we present and analyze two new fault tolerant schemes for duplicate-sensitive aggregation in sensor networks (1) *Diffused RideSharing* and (2) *Cascaded RideSharing*. These schemes make judicious use of the available path redundancy in the network to deliver a correct aggregate result to the data sink. Compared to the state of the art, our new schemes deliver a lower root mean square (RMS) error in the aggregate result, in addition to consuming much less energy and bandwidth.

Acknowledgments

This work is partially supported by NSF through grants ANI-0125704, ANI-0325353. We would like to thank Dr. Taieb Znati for suggesting the problem and for his valuable comments during the early stages of this work. We would like also to thank the database group in our department for providing us with a copy of their TiNA CSIM code, which we extended to implement our schemes.

References

[1] "CSIM simulator." http://www.mesquite.com/.
[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A Survey on Sensor Networks," *IEEE Communications Magazine*, vol. 40, pp. 102–116, Aug. 2002.

[3] A. Berfield and D. Mossé, "Efficient scheduling for sensor networks," in *IWASN*, July 2006.
[4] S. Biswas and R. Morris, "ExOR:opportunistic multi-hop routing for wireless networks," in *SIGCOMM*, 2005.
[5] Chipcon AS, "The CC1100 multi-channel RF transceiver." http://www.chipcon.com.
[6] J. Considine, F. Li, G. Kollios, and J. Byers., "Approximate aggregation techniques for sensor databases," in *ICDE*, 2004.
[7] H. Dubois-Ferrier, D. Estrin, and M. Vetterli, "Packet combining in sensor networks," in *SenSys*, 2005.
[8] D. Eckhardt and P. Steenkiste, "Measurement and analysis of the error characteristics of an in-building wireless network," in *SIGCOMM*, 1996.
[9] S. Felsner, G. Liotta, and S. K. Wismath, "Straight-line drawings on restricted integer grids in two and three dimensions," in *Revised Papers from the 9th International Symposium on Graph Drawing*, 2002.
[10] P. Flajolet and G. Martin, "Probablistic counting algorithms for data base applications," *JCSS*, vol. 31, 1985.
[11] S. Gandham, M. Dawande, and R. Prakash, "Link scheduling in sensor networks: distributed edge coloring revisited," in *Infocom*, 2005.
[12] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin, "Highly-resilient, energy-efficient multipath routing in wireless sensor networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 5, no. 4, pp. 11–25, 2001.
[13] S. Gobriel, R. Melhem, and D. Mossé, "BLAM: an energy-efficient mac layer enhancement for wireless adhoc networks," in *WCNC*, 2005.
[14] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan, "Building efficient wireless sensor networks with low-level naming," in *SOSP*, 2001.
[15] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *HICSS*, 2000.
[16] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 2–16, 2003.
[17] G. Khanna, S. Bagchi, and Y.-S. Wu, "Fault tolerant energy aware data dissemination protocol in sensor networks," in *DSN*, 2004.
[18] S. Madden, M. Franklin, J. Hellerstein, and W. Hong., "TAG: A tiny aggregation service for ad-hoc sensor networks," in *USENIX OSDI*, 2002.
[19] A. Manjhi, S. Nath, and P. Gibbons, "Tributaries and deltas: Efficient and robust aggregation in sensor network streams," in *ACM SIGMOD*, 2005.
[20] S. Mukhopadhyay, D. Panigrahi, and S. Dey, "Data aware, low cost error correction for wireless sensor networks," in *WCNC*, 2004.
[21] S. Nath, P. Gibbons, S. Seshan, and Z. Anderson, "Synopsis diffusion for robust aggregation in sensor networks," in *ACM SenSys*, 2004.
[22] M. Sartipi and F. Fekri, "Source and channel coding in wireless sensor networks using LDPC codes," in *IEEE SECON*, 2004.
[23] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis, "TiNA: a scheme for temporal coherency-aware in-network aggregation," in *MobiDE*, 2003.
[24] V. Shnayder, M. Hempstead, B. Chen, G. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in *SenSys*, 2004.
[25] F. Stann and J. Heidemann, "RMST: Reliable data transport in sensor networks," in *SNPA*, 2003.
[26] H. Tamura, K. Watanabe, M. Sengoku, and S. Shinoda, "On a new edge coloring related to multi-ho wireless networks," in *APCCAS*, 2002.
[27] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy, "PSFQ: a reliable transport protocol for wireless sensor networks," in *WSNA*, 2002.
[28] J. Zhao and R. Govindan, "Understanding packet delivery performance in dense wireless sensor networks," in *SenSys*, 2003.