



with a good model of the sensor values. In the extreme, it assumes nothing about the environment in which the sensor network is deployed. Data-driven processing still uses models, but only as optimization guidelines (more on this later), so even inaccurate models do not lead to data loss.

**Optimizing SELECT \*** With emphasis on collecting all data, it should not be surprising continuous SELECT \* is the dominant query for data-driven processing. Much of the research in our project now focuses on optimizing this query, instead of complex, database-like queries. While in a traditional database system, SELECT \* is uninteresting in terms of optimization possibilities, its continuous sensor network version is quite amenable to interesting *suppression* techniques. The naive alternative, continuous reporting, requires each node transmit its value to the base station in each time step, resulting in heavy message traffic. Suppression, on the other hand, monitors conditions within the sensor network such that data is only transmitted when readings diverge from expected (or simply default) behavior. The simplest example is *value-based temporal suppression*, where each node reports only if its value has changed beyond some threshold since last reported. This technique is quite effective when sensor values change slowly. Still, we can advance suppression further, to handle scenarios where nodes change regularly but predictably, and where changes across nodes exhibit spatial correlations. A number of suppression schemes have been proposed [2, 6, 13, 17, 22, 23, 25]. Suppression is a powerful concept, but many challenging issues remain to be addressed, e.g., how to identify missing reports as suppressions, rather than failures. These problems are addressed throughout this paper.

Although there have been some recent examples of data-driven processing in the literature [21], including by some of the designers of model-driven [2], we believe this approach has been understudied relative to the model-driven approach. The goal of this paper is to present the key issues in data-driven processing and techniques for applying them to application design. We have encountered the following issues in designing our own deployment, and generalize them for broad applicability.

- **Suppression:** This is the driving technique behind supporting continuous queries without continuous data streams. We define suppression schemes for supporting continuous queries. Most sensor networks likely exhibit temporal and spatial correlations among node readings, which are encoded within models. This behavior can be incorporated into schemes such that with limited messaging within the network, no or few messages are sent to the base station. Suppression, including incorporation of models, is covered in Section 2.
- **Message failure:** Sensor networks are prone to message drops. This is especially detrimental to suppression schemes, where non-reports are expected, but now may be the result of failures, rather than suppression. In Section 3 we discuss methods for augmenting suppression schemes against failure, and techniques for detecting failure and then inferring the actual readings and process parameters.
- **Application/communication layer interaction:** Sophisticated suppression schemes involving careful communication among multiple nodes strain the lower communication layer to provide efficient routing between such nodes. While merging these layers would greatly complicate application development, some hooks between them are necessary. In Section 4 we describe the milestone framework, for co-optimization of the layers.
- **Data representation:** Managing data produced in a sensor network and the inferred at the base station is difficult. Presenting either extreme of a view of only the data known with perfect certainty,

or a completely sanitized version of all data are both inadequate in most cases. The data is inherently probabilistic with complex correlations. These correlations, as well as details of suppression and the constraints it enforces on inference must be represented. These open problems are discussed in Section 5.

- **Role of models:** Models are discussed extensively throughout this paper. In general, models are utilized in data-driven for optimization, but not at the expense of correctness. Further, model-driven and data-driven are not static competitors; the trust and responsibility given models largely defines where along the spectrum between the approaches an application lies. This distinction can become tricky to follow, especially when discussing of failure. We review use of models in Section 6.

## 2 Suppression

Suppression is the key technique for supporting continuous queries without continuous reporting. The network, on its own volition, chooses when to push data to the base station. The intuition is if the network and base station can agree on an expected behavior, the network need only report when its readings deviate from that. The challenge is encoding expected behavior within the network, such that actual behavior can be efficiently evaluated against it.

The design space for suppression is enormous. Suppression can be utilized in a multitude of ways, even for the same query. Value-based temporal suppression, mentioned in Section 1, leverages the expectation node values are unlikely to change in a given timestep. Each node sends a message only when its value does change beyond some threshold since last reported. This scheme allows computation of SELECT \*, since the base station maintains the last value reported from each node within the threshold. If values change frequently, this degrades to continuous reporting. Spatial suppression is also possible. *Snapshot* [21] allows individual nodes to report on behalf of a local cluster, as long as nodes in the cluster have values within some threshold of the representative. This approach leverages the expectation that nearby nodes will have similar values, and minimizes the number of messages directed to the base station. If nodes do not exhibit spatial correlation, this degrades to continuous reporting.

The design space extends to more sophisticated schemes that leverage both temporal and spatial correlations. Naturally, the effectiveness of a scheme for a particular deployment depends on how well it captures the correlations existing in the deployment. In order to manage the design process, we now define a general framework for suppression schemes.

### 2.1 Suppression Scheme

**Definition** A *suppression scheme* is a set of *suppression links* deployed within the network. A link maps a suppression/reporting relationship between an *updater* node and an *observer* node. Each link synchronously maintains, with some error, a vector of quantities,  $X$ , between the updater and observer.  $X_t$  denotes the vector at time  $t$  as instantiated by the updater.  $\hat{X}_t$  denotes the vector as computed by the observer.  $\hat{X}_t$  serves as input to the observer for producing its own  $X$  vectors, to use on downstream links (observer becomes updater). Eventually, a querying node (such as the root) receives an  $\hat{X}_t$  and uses it to produce query results. Variables  $x_{t,i}$  and  $\hat{x}_{t,i}$  refer to  $i$ th quantity in the vector at the updater and observer, respectively.  $X$  may contain node readings, process parameters, and any other quantities used by the scheme. Each quantity may be generated locally at the updater (e.g. a reading taken by it) or derived from quantities received from its own upstream updaters.

For each suppression link, the user defines per-quantity precision

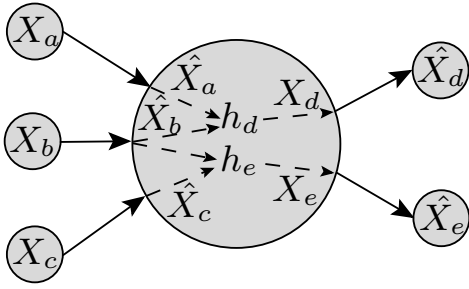


Figure 1: Suppression Scheme Graph.

bounds for each entry in  $X_t$  and  $\hat{X}_t$ . Predicate function  $g(X_t, \hat{X}_t)$  returns true if  $\hat{X}_t$  is within a prescribed error tolerance of  $X_t$ . For example, the function may be component-based, such that each  $\hat{x}_{t,i}$  must be within some range of  $x_{t,i}$ .  $X_t$  and  $\hat{X}_t$  are considered synchronized if  $g$  is true. The synchronization requirement dictates the communication necessary, if any, between the updater and observer in each timestep. The following functions encode the requirement. The updater maintains encoding function:

- $f_{enc}(X_t, X_{t-1}, \dots)$

The observer maintains decoding function:

- $f_{dec}(r_t, \hat{X}_{t-1}, \hat{X}_{t-2}, \dots)$

The updater uses  $f_{enc}$  to generate report  $r_t$  for transmission to the observer so the observer can derive  $\hat{X}_t$  within precision bounds. If no message is needed,  $r_t$  is denoted  $\perp$ , for suppression. Note because  $f_{enc}$  uses previous settings of  $X$ , it implicitly considers previous messages sent to the observer. The observer uses  $f_{dec}$  to interpret  $r_t$  and derive  $\hat{X}_t$ . If no message is received, it sets  $r_t = \perp$ . Assuming no failure,  $\hat{X}_t$  necessarily meets the precision requirements, though the observer has no way to verify this.

Note the relationships between  $f_{enc}$ ,  $r_t$  and  $X$ , and  $f_{dec}$  and  $\hat{X}$  are defined by the scheme programmer. For example,  $X$  might contain several readings and a single parameter,  $\theta$ .  $r_t$ , when not suppressed, may only consist of updates to  $\theta$ , from which all other quantities in  $\hat{X}$  are derived. We give examples in Section 2.2.

The suppression scheme is a graph of suppression links. An observer node may maintain links with one or more updaters, and may itself then become an updater for one or more observers. The node maintains a function  $h_j$  for each downstream observer to transform  $\hat{X}_t$ 's received from its updaters into its own  $X_t^{(j)}$ , to then be synchronized with a downstream observer.  $h$  functions are defined by the scheme programmer.

**Message Dependency** Each  $h_j$  also establishes a dependency between  $\hat{X}_t$ 's derived from updater messages to  $X_t^{(j)}$ .  $X_t^{(j)}$  in turn determines what is transmitted downstream for the observer to derive  $\hat{X}_t^{(j)}$ . These dependencies are *intra-node links*. We distinguish these from suppression links because they have no explicit impact on what is transmitted between nodes, but instead have impact on when that transmission may take place. Figure 1 depicts a portion of a suppression scheme graph, with the middle node, serving as an observer and then updater, enlarged. The suppression links are shown as solid arrows, while the intra-node links are shown as dashed arrows. As observer for three suppression links, the node uses  $f_{dec}$  functions to derive  $\hat{X}_a$ ,  $\hat{X}_b$  and  $\hat{X}_c$ . As updater for two suppression links, it uses  $f_{enc}$  functions to derive messages to be transmitted downstream for derivation of  $\hat{X}_d$  and  $\hat{X}_e$ . Internally, the node shows dependency on  $\hat{X}_a$  and  $\hat{X}_b$  to produce  $\hat{X}_d$ , and on  $\hat{X}_b$  and  $\hat{X}_c$  to produce  $\hat{X}_e$ .

By merging suppression and intra-node links, we derive the *h-graph*, a directed graph whose vertices are the  $h$  functions at all network nodes.

**Lemma 1.** A suppression scheme is feasible only if its *h-graph* contains no cycles.

In a particular timestep, only nodes with  $h$  functions not waiting on upstream messages may immediately transmit ( $X$  vectors from such functions are populated using input generated completely locally). If a subset of  $h$ 's at different nodes all wait on one another in a cyclical fashion, the suppression scheme waits indefinitely. The scheme completes when all terminal nodes (serving as observers, but not as updaters) receive their messages. In most query processing applications, this will be a single node, the root. The suppression scheme *supports* a particular query if  $\hat{X}$  maintained by the root is sufficient to produce the query result.

### 2.1.1 Discussion

Why go through the exercise of producing a general definition for suppression schemes, when its absence has not prevented us or others from designing schemes? We want a definition that is flexible, but exposes the important features that characterize and differentiate schemes. The definition provides a number of benefits:

- Compared with low-level programming of network messages, this abstraction makes it easier to reason about suppression algorithms. For example, it is simpler to understand the correlation an algorithm exploits, prove its correctness (both for feasibility and for supporting queries), and evaluate energy cost.
- It allows cost-based optimization in scheme design. While the design space of possible schemes remains enormous, many possibilities can be quickly eliminated due to cost. For example, a scheme with intricate spatial constraints, but higher expected cost than temporal suppression, should not be considered.
- It allows identification of optimizations general to all schemes, in contrast to application-specific optimizations. These can be presented as such and implemented once for all schemes.
- The definition can be adapted to a new application programming abstraction. Existing approaches require programmers to either plan each message by hand, or else provide only a limited set of communication patterns, such as *collection* and *dissemination* [3]. A suppression API will give programmers more control over collection, but while factoring out common tasks. We imagine, for example, setting a *cluster* abstraction by simply selecting a set of cluster members. The choice of cluster leader may be left to the API, which can make the optimal decision. The API is accompanied by cost analysis to assist in refining scheme designs.

## 2.2 Examples

We now demonstrate how the general definition can be specified to a series of existing suppression schemes.

**Temporal Schemes** We begin with value-based temporal suppression. The suppression link graph consists of link between each node and the root (i.e. a one-level link tree), where the node is an updater and the root an observer. At each node  $X$  has a single component  $x$ , its reading.  $g$  returns true if two readings are within  $\epsilon_x$  of one another. Each node maintains a local variable  $t'$ , the time at which its value was last transmitted. Its encoder function is:

$$f_{enc}(x_t, x_{t'}) = \begin{cases} x_t - x_{t'} & \text{if } |x_t - x_{t'}| > \epsilon_x \\ \perp & \text{otherwise} \end{cases}$$

The root for each link has decoder function:

$$\hat{x}_t = \begin{cases} \hat{x}_{t-1} + r_t & \text{if } r_t \neq \perp \\ \hat{x}_{t-1} & \text{if } r_t = \perp \end{cases}$$

### 2.2.1 Soil Moisture

We next examine model-encoding temporal suppression schemes for monitoring soil moisture. In our forest modeling efforts, soil

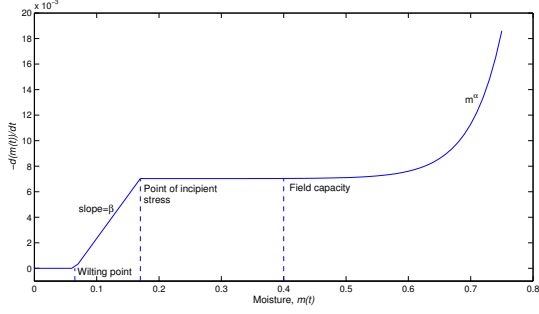


Figure 2: Soil Moisture Model

moisture is an important measurement. Moisture increases dramatically during precipitation, and then falls off either due to sub-surface run-off or transpiration by the forest. Precipitation events can be modeled as a Poisson process with amount of precipitation drawn from an exponential distribution. Modeling soil moisture is more involved. The rate of decrease in moisture after a precipitation event depends on the value of the moisture,  $m(t)$ . Figure 2 depicts the rate of decrease in  $m(t)$  as a function of  $m(t)$  for a single sensor node. We will focus on the right-most portion of the model, when moisture is above field capacity. We examine two per-node suppression schemes: PAQ [25] and exponential regression.

For both methods, the link graph is identical to that of value-based temporal. Each node maintains a link with the root. PAQ employs linear regression to have a node predict its reading each timestep using the previous three readings and parameter values,  $\alpha_t, \beta_t, \gamma_t, \eta_t$ . The prediction for  $x_t$  is as follows.

$\alpha_t(x_{t-1} - \eta_t) + \beta_t(x_{t-2} - \eta_t) + \gamma_t(x_{t-3} - \eta_t)$   
 $\alpha, \beta,$  and  $\gamma$  are derived by regression and  $\eta$  is the mean value of the past readings. These parameters evolve over time, necessitating their subscripts. The root knows the parameters and simultaneously makes the same prediction. If the node's prediction is within  $\epsilon_x$  of its reading, it suppresses. Otherwise, it has the option of transmitting the reading as an outlier, or revising and transmitting the parameters. The details underlying this decision are in [25] and are local to each node. For our purposes, we abstract these away by assuming the node has boolean functions `modelRefit` and `outlier`. If  $|x_t - x_{t-1}| > \epsilon_x$ , one and only one of these returns true (otherwise, neither are true). If `modelRefit` returns true, the function has also updated  $\alpha_t, \beta_t, \gamma_t$  and  $\eta_t$ . Note `modelRefit` and `outlier` must mimic the prediction made at the root, using the previous three predictions ( $\hat{x}_{t-1}, \hat{x}_{t-2}, \hat{x}_{t-3}$ ) rather than the actual readings.

We now place PAQ in our framework.  $X_t = [x_t, \alpha_t, \beta_t, \gamma_t, \eta_t]$ .  $g$  dictates a separate  $\epsilon$  for each component of  $X$ .  $\epsilon_x$  is user-defined.  $\epsilon_\alpha, \epsilon_\beta, \epsilon_\gamma$  and  $\epsilon_\eta$  are all set to 0 (i.e. the root is notified of any updates). Each node has encoder function:

$$f_{enc} = \begin{cases} \alpha_t, \beta_t, \gamma_t, \eta_t & \text{if (modelRefit)} \\ x_t & \text{if (outlier)} \\ \perp & \text{otherwise} \end{cases}$$

The root has decoder function:

$$\hat{\alpha}_t, \hat{\beta}_t, \hat{\gamma}_t, \hat{\eta}_t \leftarrow \begin{cases} \alpha_t, \beta_t, \gamma_t, \eta_t & \text{if } r_t = [\alpha_t, \beta_t, \gamma_t, \eta_t] \\ \hat{\alpha}_{t-1}, \hat{\beta}_{t-1}, \hat{\gamma}_{t-1}, \hat{\eta}_{t-1} & \text{otherwise} \end{cases}$$

$$\hat{x}_t \leftarrow \begin{cases} x_t & \text{if } r_t = x_t \\ \hat{\alpha}_t(\hat{x}_{t-1} - \hat{\eta}_t) + \hat{\beta}_t(\hat{x}_{t-2} - \hat{\eta}_t) + \hat{\gamma}_t(\hat{x}_{t-3} - \hat{\eta}_t) & \text{otherwise} \end{cases}$$

We next look at exponential regression. Each node has a prediction model for suppression:  $x_t = \alpha_t x_{t-1} + \beta_t$ .  $X_t = [x_t, \alpha_t, \beta_t]$ . Like PAQ,  $\epsilon_x$  is user-defined, while  $\epsilon_\alpha$  and  $\epsilon_\beta$  are 0. Each node has encoder function:

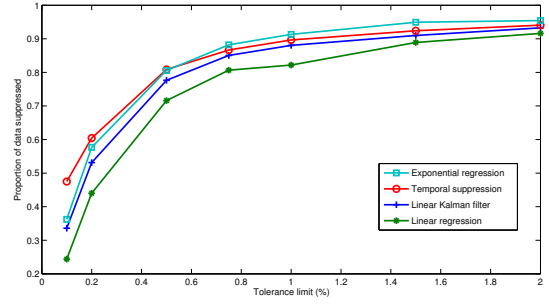


Figure 3: Suppression Results

$$f_{enc} = \begin{cases} \alpha_t, \beta_t & \text{if (modelRefit)} \\ x_t & \text{if (outlier)} \\ \perp & \text{otherwise} \end{cases}$$

The root has decoder function:

$$\hat{\alpha}_t, \hat{\beta}_t \leftarrow \begin{cases} \alpha_t, \beta_t & \text{if } r_t = [\alpha_t, \beta_t] \\ \hat{\alpha}_{t-1}, \hat{\beta}_{t-1} & \text{otherwise} \end{cases}$$

$$\hat{x}_t \leftarrow \begin{cases} x_t & \text{if } r_t = x_t \\ \hat{\alpha}_t(\hat{x}_{t-1}) + \hat{\beta}_t & \text{otherwise} \end{cases}$$

Framing both of these approaches as suppression schemes makes for an elegant comparison. First, they both use local suppression at each node. This commonality is exposed in their suppression link graphs, which are identical. The differences are the encoding and decoding functions. Given the probability of how often each component of  $X$  is suppressed, it is straightforward to predict the cost of each scheme.

We now return to Figure 2. Moisture drops exponentially when moisture is high (on the right side of the graph). We expect a suppression scheme aware of this to have an advantage over those that do not. This is a good example of the role of models in data-driven acquisition. All of the discussed schemes produce readings within  $\epsilon_x$  of the actual. But because exponential regression best models the data, it should have the most accurate predictions and the most suppression. In Figure 3, we compare suppression rates for value-based, linear regression (PAQ), exponential regression, and a Kalman filter variation [17] using simulated moisture data. Error tolerance is varied on the x-axis and corresponding suppression rate is plotted on the y-axis. We see exponential regression indeed achieves the best suppression rate, though the improvement over the others is not dramatic. We are investigating these types of comparisons further.

### 2.2.2 Spatio-temporal Suppression

We next look at a suppression scheme that exploits not just temporal correlations at individual nodes, but spatio-temporal correlations between multiple nodes. This requires monitoring spatial conditions in-network, so the suppression scheme will not consist solely of links between each node and the root.

**Conch** The scheme is one we have previously designed, *Conch* [23]. We examine a value-based version of it, making it a spatio-temporal equivalent of value-based temporal suppression. Note we employ a different version of error tolerance than published in [23] that better mirrors the prior temporal schemes (we discuss both shortly). The main idea is to monitor network edges connecting neighboring nodes; specifically, we monitor the difference in value between each edge's endpoint nodes. The root maintains the current difference across each such edge. The intuition is to choose edges whose nodes are correlated; when their values change, they will change synchronously by similar amounts, and the difference will not change and be suppressed. We next define monitoring of edges in our suppression framework, and then describe how to build

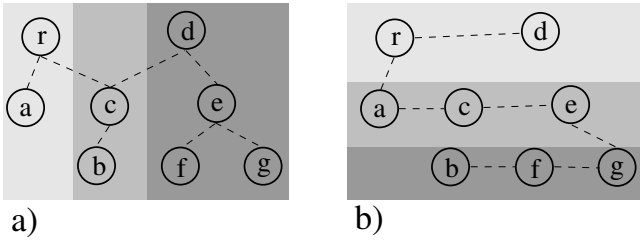


Figure 4: Conch Example

a suppression scheme to support SELECT \* from these.

Monitoring an edge requires communication, and thus a suppression link, between its endpoint nodes  $u_1$  and  $u_2$ , and then another link between  $u_2$  and the root.  $g$  at  $u_1$  returns true if values are within  $\epsilon_x$  of one another.  $X$  at  $u_1$  contains only its reading,  $x$ .  $u_1$  maintains a local variable,  $t'$ , the time in which it last transmitted to  $u_2$ .  $u_1$  has encoder function:

$$f_{enc} = \begin{cases} x_t - x_{t'} & \text{if } |x_t - x_{t'}| > \epsilon \\ \perp & \text{otherwise} \end{cases}$$

$u_2$  has decoder function:

$$\hat{x}_t = \begin{cases} \hat{x}_{t-1} + r_t & \text{if } r_t \neq \perp \\ \hat{x}_{t-1} & \text{if } r_t = \perp \end{cases}$$

Note these functions are the same as used by the node-root link for temporal suppression.

$u_2$  measures its own reading,  $y$ .  $X$  at  $u_2$  contains only the difference in readings across the edge,  $\Delta$ . To produce this,  $u_2$  has function  $h_\Delta = y_t - \hat{x}_t$ .  $g$  returns true if two  $\Delta$ 's are within  $\epsilon_\Delta$  of one another.  $u_2$  maintains a local variable,  $t''$ , the time in which it last transmitted to the root.  $u_2$  has encoder function:

$$f_{enc} = \begin{cases} \Delta_t - \Delta_{t''} & \text{if } |\Delta_t - \Delta_{t''}| > \epsilon \\ \perp & \text{otherwise} \end{cases}$$

The root has decoder function:

$$\hat{\Delta}_t = \begin{cases} \hat{\Delta}_{t-1} + r_t & \text{if } r_t \neq \perp \\ \hat{\Delta}_{t-1} & \text{if } r_t = \perp \end{cases}$$

Though more details are available in [23], we now briefly explain how to support SELECT \* with edge monitoring. In one special case of Conch, we monitor the root with temporal suppression and monitor a set of edges such that they form a spanning tree over the network. If a single node serves as  $u_2$  for multiple edges, its  $X$  contains a  $\Delta$  for each. Its encoder function suppresses  $\Delta$ 's that have not changed by more than  $\epsilon_\Delta$  since last transmitted, and reports those that have.

Two example Conch spanning trees are shown in Figure 4. For each edge, the root knows the current difference between its vertices within  $\epsilon_\Delta$ . To derive any node's value, the root finds a path of edges in the spanning tree from itself to the node (by definition of spanning tree, exactly one path must exist). The root then starts with its own value and modifies it by each edge difference in the path, a process called *chaining*. Many different spanning trees can be built over a single network. As we see from the scheme declaration, the cost of monitoring a particular edge is tied to how often  $x$  changes by more than  $\epsilon_x$  and the cost to communicate from  $u_1$  to  $u_2$ , and how often  $\Delta$  changes by more than  $\epsilon_\Delta$ , and the cost to communicate from  $u_2$  to the root. If two nodes' values demonstrate high correlation, their edge is a good candidate for monitoring; it will seldom need to report to the root. Suppose in Figure 4, each shaded region exhibits a particular behavior, such that all nodes in a single region are correlated. Any edge between nodes in the same region will never report to the root, while edges between nodes in different regions will frequently report. The goal in building the spanning tree is to select as few edges that cross regions as possible. In both examples, each with three regions, we are forced to choose two such edges. Imagine we trade the spanning trees, such

that the tree in 4a is assigned to the environment in 4b, and vice-versa. It is easy to see reporting costs will increase in both cases.

**Error** We see interesting implications for suppression parameters  $\epsilon_x$  and  $\epsilon_\Delta$ . The suppression link graph has hierarchy of depth two. At  $u_2$ ,  $\hat{x}_t$  can diverge from  $x_t$  by as much as  $\epsilon_x$ . Thus,  $\Delta$  may diverge from  $y_t - x_t$  by  $\epsilon_x$ . This error accumulates at the root:  $\hat{\Delta}_t$  may diverge from  $y_t - x_t$  by  $\epsilon_x + \epsilon_\Delta$ . The same problem carries over to chaining. For each edge involved in chaining to a particular node, error accumulates. For a chain of length  $l$ , the computed node's value has error up to  $l(\epsilon_x + \epsilon_\Delta)$ . To achieve the same accuracy as temporal suppression, it is necessary to use lower  $\epsilon$  values, and likely different  $\epsilon$ 's for each edge. It is important to understand the error implications of a suppression scheme. The suppression link graph view of the scheme exposes this.

Original Conch suppression works as follows. Nodes monitor not  $x$ , but the discrete quantity,  $x' = \lfloor x/\epsilon \rfloor$ , with error bound of 0. Differences are computed on discrete quantities, also suppressed with error bound 0. We estimate  $\hat{x}_t$  as  $x'_t \epsilon$ , and can guarantee  $\hat{x}_t \leq x_t < \hat{x}_t + \epsilon$ . The advantage of this approach is it makes chaining less susceptible to error accumulation. The disadvantage is this discretization is somewhat artificial; small changes in actual value that happen to cross a discretization step must be reported.

### 2.3 Toward an Optimization Framework

We are building toward a framework for suppression optimization. The first component of this is the suppression scheme definition. As mentioned, this allows us to manage the large design space of suppression schemes, and to evaluate cost for each. We foresee two levels of optimization. The higher level is the compile-time decision of what scheme should be deployed in the network, such as temporal suppression and, as we have seen with soil moisture, what models can be monitored within that scheme.

The lower level is at run-time. The network itself makes dynamic adjustments to suppression. For PAQ, this involves deciding when a sequence of outliers actually signifies a change in process parameters (on which suppression will now be based). The moisture model contains three distinct components. An effective suppression scheme must encode all of them and know when to shift between. One possibility is to set choice of model as a state parameter in  $X$ , reported only when the node shifts, and suppressed otherwise. In these cases, the ability to modify suppression is pre-compiled within the network, and need only be invoked by nodes at run-time.

A final question is how and when to make externally initiated changes to the suppression scheme. For example, the base station may contain a rain sensor, making it better qualified than individual nodes to decide whether increased moisture readings are outliers or a shift to the high-moisture component of the model. Further, the base station collects a great deal more information than any particular node. Any sweeping changes to the scheme, which likely come with high energy cost, should be made by the base station.

**End Goal** We have extensively discussed the use of suppression, design of suppression scheme, and integration of models into these. Beside our omnipresent goal of energy-efficiency, it is easy to lose sight of what we hope to achieve. To that end, we have the following succinct vision for suppression:

1. No messages reach the base station reporting expected behavior.
2. For each unexpected phenomenon affecting the network, only one message reaches the base station.

If all nodes are affected similarly by the same event, they should detect this in-network and not independently report it to the base station. Nevertheless, this goal must be tempered by the in-network

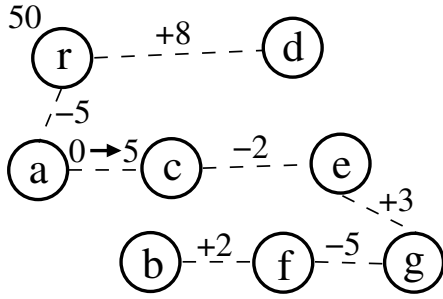


Figure 5: Conch Failure

cost to coordinate the single report. In the end, this vision is a guideline to motivate suppression scheme design rather than something to be achieved in practice.

### 3 Message Failure

Failure is a serious problem in sensor networks, with message loss common. Experimental results have shown transmission loss rates to be at times 50% and higher, with congestion blamed for the majority of these [16]. Congestion causes message interference, where multiple messages are sent in the same air space, causing all of them to be corrupted. When message buffers at nodes fill to capacity, nodes must either drop messages as they arrive, or delete messages from their buffers; in either case, messages are lost. These findings were from a high-traffic network running a continuous reporting application. To some degree, suppression should naturally mitigate these problems. Other work blames failure on functioning, but unreliable, links, for which multiple transmissions are likely needed to achieve successful delivery [26, 27]. In outdoor deployments, environmental interference can certainly cause message loss. In ours, for example, connectivity degrades during the summer, when there is more foliage. Failure not only occurs at the message level, but at the bit level within each message. For now we assume corrupted bits are corrected (message is successful) or cannot be (message fails) at a lower level.

Failures are detrimental to suppression schemes. In the absence of failure, observers assume non-reporting suppression links have not transmitted a message, but suppressed. Failure creates ambiguity: now an observer must consider the possibility its updater did transmit a report, but that it was lost. For value-based temporal suppression, a single lost report from a node results in the node’s value being mis-set by the base station. For more complex schemes, the impact may be more widespread. Figure 5 depicts a Conch example, labeled with the latest reported value for each spanning tree edge, with  $r$  as the temporally monitored root. In the current timestep, the edge  $a \rightarrow c$  reports its difference rising from 0 to 5. Suppose this report is lost. It is easy to see the base station will mis-calculate the values of nodes  $c, e, g, f$  and  $b$  in chaining. The single failure affects all node values computed using  $a \rightarrow c$ , and continues to affect them at least until  $a \rightarrow c$  changes again and attempts to report. Without further effort, the base station has no way to differentiate suppressions and failures and remedy this.

**Coping at the Network Layer** The most straightforward strategy for coping with failure is to eliminate it. At the medium-access (MAC) layer, we can require each message along a single hop from sender node to receiver be followed with an acknowledgment message from the receiver. If no acknowledgment is received after some time, the sender sends the original message again. After some number of attempts, however, the sender must give up. At that point, the MAC layer may return the transmission back to the communication layer, and request an alternative path be used.

Eventually, the sender may run out of paths and attempts at each and give up entirely. Fundamentally, there is no way to fully eliminate the chance of failure.

**Coping Implicitly** Some applications assume there is enough naturally occurring redundancy from temporal and spatial correlation in data to compensate for failure. In continuous reporting applications, if values are received from most nodes, missing ones can be filled in with neighbors’ values. If a node’s value is missing in one timestep, it can be filled in with the average of its values in adjacent timesteps. The suppression algorithm Ken [2], which tries to maintain an accurate model of the network at all times, assumes any mistakes due to failure will eventually be corrected when reports are received. They use heartbeat messages to ensure mistakes do not last indefinitely.

**Our Approach** We have advocated heavily for using suppression specifically because it removes redundancy in network reporting, saving energy. This raises important questions. If redundancy is necessary to compensate for failure, is there any point to using suppression? Or in coping with failure, will we revert suppression schemes back toward continuous reporting?

We will not revert. Suppression lets us remove redundancy so we can add it back in a controlled fashion. There exists a fundamental trade-off between redundancy cost and benefit. If we rely on natural redundancy, we have no control over this trade-off. By explicitly adding redundancy we have very flexible control. If certain suppression links are very reliable, for example, we need not add much redundancy to them or on their behalf. In general, redundancy is a component of suppression scheme design.

### 3.1 Application-Level Redundancy

The goal in application-level redundancy is not to reduce failure (as with MAC layer re-transmission), but to make applications robust to it. The base station’s ability to produce a correct (within user-defined error) query result does not hinge on any particular reports successfully transmitting. This has a number of advantages over re-transmission (though the two can be applied in concert). First, if many re-transmissions are needed on average, that strategy may become quite expensive. The application redundancy we propose can often be added onto existing messages, saving on overhead costs. Second, programming nodes at the application-level is arguably easier than at the lower MAC and communication layers.

We rely on both redundancy and models of network behavior to produce a query result. This is clearly a blend of data-driven (redundancy) and model-driven (model) techniques; we discuss how to tune between these.

### 3.2 Examples

**Temporal Suppression** We begin with value-based temporal suppression. A node only transmits when its value differs by more than  $\epsilon$  since its last transmission. We make a subtle tweak to minimize the impact of failure. Until now we have had the node transmit the difference from the last transmission, potentially saving a small number of bits versus transmitting the new value itself. The difference is used to update the node’s value stored at the base station. With this approach, however, failures accumulate on top of one another and, even when interrupted by successful transmissions, perpetuate indefinitely. There is never a re-calibration that tells the base station the correct value. To prevent this, on each transmission we report the new value itself. This means we at least learn the correct value on each successful transmission, eliminating the effect of previous failures on future derivations. The sacrifice of a few extra bits is likely not significant. This illustrates the types of modifications we must make to deal with failure.

We now examine four versions of value-based temporal suppression. The first is the standard version, while the latter three add redundancy to their payloads at increasing cost.

- Standard
- Counter
- Timestamp
- History

Regardless of version, as before, a node only transmits if its value differs by more than  $\epsilon$  since last transmitted. With each transmission, however, the application payload differs. This means the number of reports, and thus the total overhead paid on behalf of the lower network layers (MAC and communication), is identical. We prefer this approach to sending messages at more timesteps and paying more overhead. The root uses the received messages to fill in the set of the node’s readings over time,  $V$  ( $v_t$  denotes the reading at time  $t$ ). If it does not know the exact reading for a particular time, it tries to fill in one of two special symbols:  $s$  for suppression or  $f$  for failure. Combined with  $\epsilon$ , these place constraints on the possible actual values to be used later in producing results. If the base station cannot distinguish between  $s$  and  $f$ , it fills in **na**, for “not available.”

**Differentiating the Approaches** Standard simply transmits the node’s reading. If the base station receives a report at time  $t$ , it sets  $v_t$  to the received reading. If it does not, it must set  $v_t$  to **na**.

A node running Counter increments a local counter on each report sent to the root and adds it to the standard message. Suppose the root receives reports with none between at  $t$  and  $t + z + 1$ , but with non-consecutive counter values  $c$  and  $c + f + 1$ . The root detects  $f$  failures and  $z - f$  suppressions have occurred in  $z$  timesteps. It can estimate a recent failure rate of  $f/z$  from the node, and can enumerate  $\binom{z}{f}$  possible suppression/failure scenarios to fill in the gap between reports. Using knowledge of a model and reports from other nodes, some permutations may be more likely than others, letting us express the actual values in  $V$  with some likelihood, or generate samples of them.

A node running Timestamp includes in its reports a list of timesteps of the last  $n$  times in which it transmitted, ordered from most recent going backward. The root applies this list to  $V$  as follows:

- Loop through each time  $k$  from current time  $t$  back to the earliest time listed in the timestamp list.
- If  $k$  is listed in the timestamps and  $v_k$  is currently set to **na**, set  $v_k$  to  $f$ .
- Else if  $k$  is not listed in the timestamps, set  $v_k$  to  $s$ .

This playback fills in timestamps as  $s$  or  $f$ , but only backward to the earliest listed timestamp. Prior to that, there is no way to distinguish failures from suppressions. The greater  $n$ , the more consecutive failures must occur for timestamps to be left as **na**. We expect a small constant  $n$  to be effective, and not add much cost compared to Counter. Knowing the exact positions of  $s$ ’s and  $f$ ’s lets us place bounded constraints on the actual values in  $V$ . For example, if we know  $v_t$  and identify  $v_{t+1}$  as a suppression,  $v_{t+1}$  must be within  $\{v_t - \epsilon, v_t + \epsilon\}$ .

History, finally, transmits along with the last  $n$  timestamps, the readings taken at them. Given the same  $n$ , for each  $v_t$  Timestamp identifies as  $f$ , History fills in the actual reading. If  $n$  is high enough to eliminate all **na**’s in  $V$ , this is sufficient to bound all readings within  $\epsilon$ .

It is easy to see as the versions increase in payload size and therefore consume more energy in transmission, we also increase our knowledge, or certainty, about the values in  $V$ . We illustrate this trade-off by comparing Standard and Timestamp experimentally.

Both schemes have  $\epsilon$  set to 0.3 and are aware raw data is generated according to an auto-regressive(1) process. We examine a series of four consecutive timesteps, where only the endpoints are reported at the base station. The series produced by each scheme are:

- Actual:  $\{-2.5, -3.5, -3.7, -2.7\}$
- Standard:  $\{-2.5, \text{na}, \text{na}, -2.7\}$
- Timestamp:  $\{-2.5, f, s, -2.7\}$

We have constructed 2-D joint scatter plots to show possible reconstruction combinations of the two missing values. Figure 6 shows a joint scatter plot of samples inferred using Standard, while Figure 7 shows samples from Timestamp. We observe Timestamp establishes stronger bounds, eliminating many possible reconstructions. Qualitatively, we see two scenarios: one where the second value rises above -2.2 and one where it falls beneath -2.8 (the latter scenario is the correct one). In either case, because the third value is a suppression, each of its samples must be within 0.3 of the sample generated for the second value.

**Conch** We have discussed a variety of ways to add redundancy to temporal suppression. For Conch, we again report not the changes in difference for each monitored edge, but the actual difference, to prevent failures from compounding over time. We sketch out one idea for adding redundancy to Conch that adds additional monitored edges. For example, in Figure 5, we can add edge  $r \rightarrow c$  to provide another path from  $r$  to each of  $c, e, g, f$  and  $b$ . The edges are used to construct linear constraints on the true edge differences. For this example,  $(r \rightarrow a) + (a \rightarrow c) = (r \rightarrow c)$ . If this constraint does not hold, at least one of the three edges must have failed to report a change.

While detecting failure is important, it still leaves the problem of inferring the true node values. The more redundant edges added, the more evidence there is with which to produce the correct values, but of course at an added cost. It is not clear how to best add redundancy. This likely involves a combination of extra edges and the per-report additions used for temporal suppression. What edges should be added? Ideally, there should be multiple independent paths to each node. Further, edges meant to “cover” for one another must not have correlated failure, or else they provide no benefit and only extra cost.

### 3.3 Inference

The inference problem is to determine the actual sequence of values, or a sample of them, as in the examples depicted in Figures 6 and 7. This is quite complex and here we simply sketch the input, output, and strategies. First, the temporal suppression examples show how the suppression scheme, notably  $\epsilon$ , and redundancy provide constraints on possible reconstructions of the raw values. Using constraints based on evidence drawn from the network is a data-driven concept. We can also utilize models of node behavior, such as the AR(1) used in the examples. Even when values cannot be hard-bounded, such as with Standard, AR(1) dictates the actual values are unlikely to deviate far from timestep to timestep. This is visible in Figure 6, where the distributions for  $x_2$  and  $x_3$  are conditioned on the known values  $x_1$  and  $x_4$ . Using a model in this way is clearly a model-driven concept; the samples are not derived purely using evidence drawn from the network. It is possible to control model reliance. We can diminish its importance relative to data-driven evidence by, for example, lowering  $\epsilon$  or increasing redundancy. The samples are then influenced more by constraints than the model, but at increased cost. We see another fundamental trade-off, then, between energy cost and reliance on model. Aside from reconstructing the actual readings, we can also try to learn the model process parameters. In our deployment, where the end goal

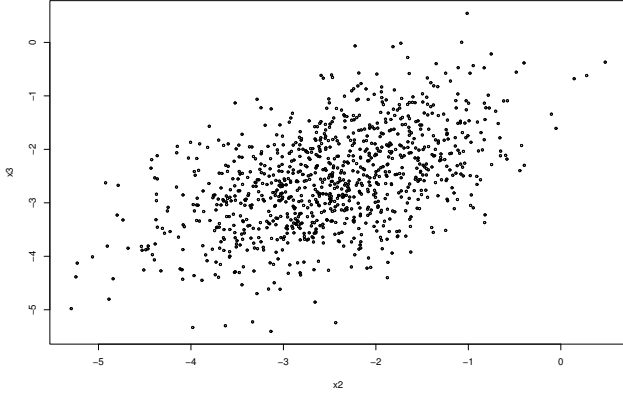


Figure 6: Missing values are na.

is to construct models of forest growth, this may be preferable.

We use a Bayesian approach to infer the raw values or process parameters. This integrates all knowledge into inference, whether encoded in the suppression scheme, reported from the network, or from prior belief of the process parameters. This approach has the attractive property of giving confidence measurements for the inferred values and parameters, as opposed to simply mean values. This is especially important for measuring scheme cost and trust of models. It is easy to see the certainty benefit gained through a particular cost and/or trust level. Confidence is also crucial to post-processing efforts.

**Post-Processing** The confidence given by inference may leave the user unsatisfied. In fact, if the user is always satisfied, the scheme should arguably be tuned to lower confidence, and lower energy cost. Post-processing directs gathering of additional evidence from the network to raise confidence. Suppose the user demands all raw values for a node be inferred within a size  $r$  range with 95% confidence, and this is not achieved initially. Further suppose the node archives its readings locally until its limited memory forces evictions. For a limited time after inference, the root can actively query the node to retrieve historical values. One obvious strategy is to fetch all values for which inference confidence is too low. More efficient solutions are possible, however. The root may be able to acquire a limited set of values and contribute it as evidence to infer all values, raising all confidence levels to sufficient levels. The optimization problem is to find the minimum acquisition cost subset of values that augments inference to derive all values with sufficient confidence.

## 4 Interacting with the Communication Layer

The concepts we have discussed to this point focus mainly on the application layer. There are many opportunities for optimizing interaction between multiple layers, but danger in making coding tasks too difficult. Nevertheless, the energy-efficiency of many of our techniques depends on behavior at lower levels. In particular, we now examine the communication, or routing, costs.

In Section 2 we examine several suppression schemes. None of their suppression link graphs are very taxing to the communication layer. Temporal suppression monitoring takes place locally at each individual node. *Conch* edges are monitored between neighboring nodes. When these nodes or edges generate reports, the reports are turned over to the communication layer, which transmits them to the root through whatever path it finds most efficient. These suppression schemes do not request much of the communication layer, though we see some conflict emerging in *Conch*. An edge

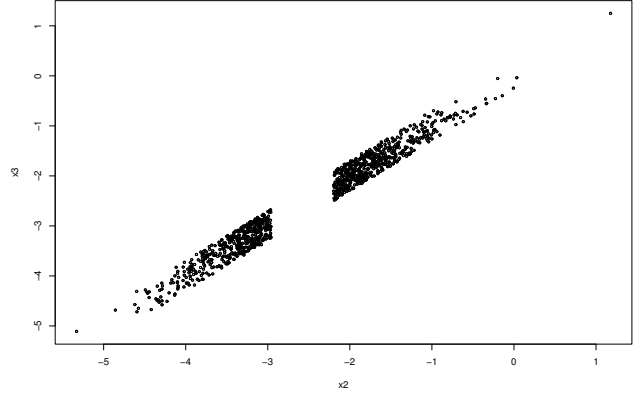


Figure 7: Missing values are  $f, s$ .

is eligible for monitoring because its vertex nodes are neighbors; they can communicate with one another directly. In the future, the connection may degrade, and the communication layer will then need to find an alternate, detour, route between the nodes, making the edge less efficient to monitor. *Conch* must adjust by removing that suppression link and adding replacement edges.

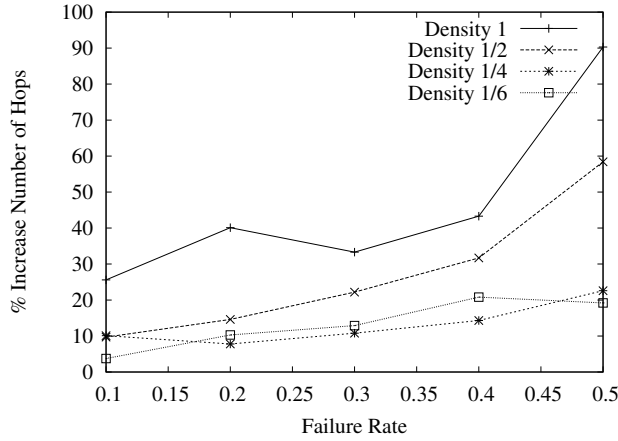
**Example** We plan to design applications with more complex spatial reporting that require more than single-hop routing and so will place a greater burden on communication. Consider the problem of an in-network join query on three nodes,  $u_a, u_b$  and  $u_c$ , producing readings  $x_a, x_b$  and  $x_c$ . The join produces a result if all three nodes' values simultaneously exceed some value,  $p$ . This can easily be modified to a window query to avoid synchrony issues.

The query need only notify the root when the result is non-empty, i.e. when all three nodes' values exceed  $p$ ; otherwise the network can suppress. Each node individually monitors its value against  $p$  and suppresses if its selection predicate is not met. But can any suppression be done in-network on the join? This is only possible if it can be determined in-network whether or not all three of the nodes pass. Once a node generates a value that passes its predicate, the communication layer takes over sending a report of this to the root. The application layer, oblivious to this routing, does not know whether reports from each node, if generated, will converge en route to the root. The application has no choice but to assume all three nodes have generated reports, but convergence will not occur until the root.

Suppose instead each node reports to some shared intermediate node,  $u_m$ , located close to all three of them. If  $u_m$  receives reports from all three nodes, it transmits a report to the root. Otherwise, it can safely suppress. The use of  $u_m$  is particularly beneficial if, say,  $u_a$  and  $u_b$  often pass their selection conditions, but  $u_c$  does not, and  $u_m$  is located close to  $u_a$  and  $u_b$ . We can frame our options as suppression schemes. In the first, the link graph consists of three links, between each node and the root. In the second, the graph consists of links between each node and  $u_m$ , and a fourth link between  $u_m$  and the root.

**Routing Implications** Setting an intermediate node,  $u_m$ , requires the communication layer support point-to-point routing. Such schemes exist [12, 20], but the sensonet operating system TinyOS [1] currently most easily handles many-to-one communication, where all messages are directed to the root, and neighbor communication, where all messages are single-hop.

Interspersing  $u_m$  as an intermediate node can only increase the path length between each of  $u_a, u_b$ , and  $u_c$ , and the root. The triangle inequality principle shows the shortest combined distance



**Figure 8: Impact of Failure on Routes**

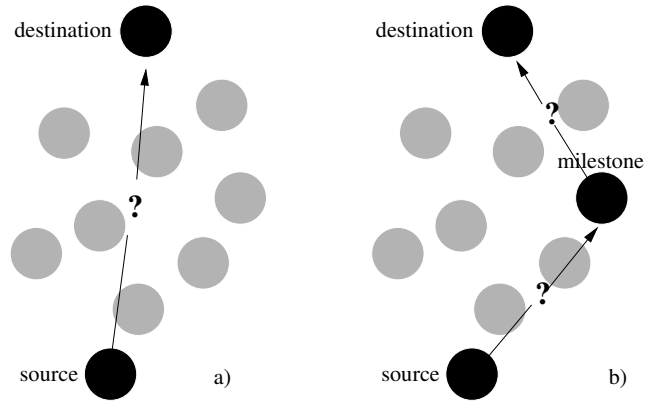
between  $u_a$  and  $u_m$ , and  $u_m$  and the root, is greater or equal to the shortest distance between  $u_a$  and the root. Shortest path here not only refers to number of hops, but also the number of transmissions over those hops. An edge with poor connectivity necessitates a greater expected number of re-transmissions than one with reliable connectivity. Even if  $u_m$  initially lies within the shortest path between  $u_a$  and the root and seems to add no cost, reliabilities may change over time such that the shortest path no longer contains  $u_m$ .

This intuition is tested in Figure 8. We simulate a network by setting node locations and constructing edges between all pairs of nodes within some distance of one another. We then choose arbitrary pairs of sources and destination nodes. For each pair, we determine the shortest path between them. We then set certain nodes along each path as intermediate nodes that must be traversed, varying the density with which such nodes are imposed. A density of 1, for example, means each node along the path is designated as intermediate. A density of 1/6 means every sixth node is intermediate. As long as all edges function, there is no additional routing cost to traverse the intermediates; they all lie along the shortest path. We then randomly fail edges in the network with some probability. Failure rate is varied on the x-axis; for each rate, all edges fail with that probability. With edges now missing, we again compute the shortest path between each pair, both with no intermediates, and with the imposed intermediates as dictated by density. For each combination of density and edge failure rate, we plot the average percentage increase in number of hops traversed between source and destination when including the intermediates versus not.

As expected, the higher the density and the higher the failure rate, the greater the increase in number of hops. The benefit to in-network processing must be large in these cases to counter the extra hops. On the other hand, when failure rate is low, we can afford to impose more on routing, even if the processing benefit is only slight. The goal is to find the optimal balance.

## 4.1 Milestone

We now formalize the trade-off between application processing and routing burden. We label intermediate nodes *milestones*, from the idea the communication layer has complete freedom in routing messages, but must deliver them to the milestones. For a source and destination (such as the root) pair, by declaring one or more milestones we ensure messages generated at the source will traverse the milestones en route to the destination. This is illustrated in Figure 9, where the gray nodes signify both the freedom given the communication layer between milestones, as well as the application's ignorance to what routes will be chosen. The application can count on



**Figure 9: Milestone Example**

messages arriving at the milestones and plan in-network processing at them. The power of the milestone approach is it serves as a link between the application and communication layers through which we can balance costs, without requiring either know the details of the other. At one extreme, we can declare every node a milestone, forcing a static routing tree, but having maximal opportunities for processing. At the other, we can declare no milestones, giving the communication layer total freedom, but having no opportunities for processing between sources and destinations.

### 4.1.1 Optimization

The milestone optimization problem bridges the communication and application layers. The former provides a collection of  $n^2$  virtual edges between each pair of network nodes, coupled with their base costs. An edge's base cost roughly corresponds to the number transmissions needed to traverse it, summing both multiple hops, and number of re-transmissions over the same hop. Thus, an unreliable edge between physically nearby nodes may have the same base cost as a reliable edge between more distant nodes.

The application layer provides a set of source and destination node pairs. Each pair represents a delivery requirement that must be fulfilled by the application. For each delivery pair, one or more adjacent virtual edges must be selected whose endpoints connect the source and destination. For a given selection of virtual edges, it is possible to compute the message cost across each edge. This is based on the nature of the application such as, for example, whether multiple source values converging at a virtual edge raise or lower message cost. The optimization goal is to minimize total cost. This is computed as the sum of the products over all utilized virtual edges of base cost and message cost. This achieves a clean interaction between the two layers. Both contribute input to co-optimization, but neither need know what routing protocol or application algorithm, respectively, supplies those inputs.

**In-Network Join** We now describe the optimization for our in-network join. The application supplies three source-destination pairs. The sources are  $u_a, u_b$ , and  $u_c$ . The destinations for all three are the root. A source-to-root path must be supplied for each pair. The message cost for each utilized virtual edge depends on the number of sources converging at its first endpoint node. A virtual edge originating at one of the sources has average message cost equal to the probability  $x > p$ . For a virtual edge shared by multiple sources, the message cost is equal to the product of the converging sources' probabilities (assuming independence). Thus, we can reduce message cost by having the source nodes share virtual edges. To achieve minimum overall cost, however, this effort must be tempered by the base costs of the chosen edges. The optimal setting is the choice of virtual edges that provides paths for all

source-destination pairs and minimizes total cost.

### 4.1.2 Suppression Scheme

At this point, milestone should seem closely related to our discussion of suppression scheme design from Section 2. In fact, milestone is general enough to evaluate suppression scheme cost and therefore aid in scheme selection. The suppression link graph is a set of virtual edges. Each node serving as an observer (i.e. the non-leaves in the suppression link graph) is a milestone. We need not worry about how reports are routed between updaters and observers, but only what such transmissions cost.

We expect our intuition developed in this section to carry over to suppression scheme design. Programmers may be tempted to translate a sophisticated model of network correlation into an intricate suppression scheme with a deep hierarchy of suppression links, and therefore many milestones. The communication burden for such a scheme, however, may outweigh the suppression benefit. The schemes which best balance these two costs will win out.

## 5 Data Representation and Storage

Following discussion on suppression and failure, it should be evident presenting query results to the user is non-trivial. One option is to only show the raw data collected at the base station, in whatever form it be. This would make us quite derelict in our duty to supply query results, especially if we produce high levels of suppression but do not explain what all the missing data means. On the other hand, due to the inherent uncertainty in sensornets, it is irresponsible to simply present a standard, fully populated relational database as though data has been manufactured locally without error.

Data collection is the focus of the data-driven approach, and the details of this collection must be stored and presented. Storing metadata is important, and includes what suppression scheme is in use, as well as the suppression parameters, such as  $\epsilon$ . What should be presented as data itself? That depends on the intended audience. If the audience is not known, the best policy might be to store everything, including samples for each raw value and process parameter. The user can clean the data and store it in a relational database as they see fit. *ESP* [18] is a SQL-like language for directing cleaning that may be suitable for this task. If the user is non-technical, it may be adequate to present a cleaned view of the data, perhaps using MauveDB [11], for example. We believe applications designed for non-technical users should also build in expectations for confidence levels that are satisfied automatically.

Finally, users such as our ecologist collaborators may not want an automatically cleaned view of the data, whether they direct it or not. They must know exactly what is known for sure from the suppression scheme and what is estimated from the model. This is key to accurately exposing the blend of data-driven and model-driven techniques used. Consider the samples for missing values shown in Figures 6 and 7. When a missing value following a report is known to be a failure, we obtain hard bounds on its actual value. When nothing is known about the missing value, we lose the hard bounds but can still concentrate the samples as not having moved far from the previous reported one. This is the type of subtle distinction that must be efficiently encoded, and is especially important when considering the end goal of building models. If we sanitize the data using a prior model, and then build a model from it, the new model will certainly be biased toward the prior.

The challenge of representing probabilistic data in a relational database is an active research area, and we are adding fuel to the fire. The suppression scheme represents a new type of metadata. Our probabilistic data contains temporal and spatial correlations. These must not be lost in storing samples. In typical databases

there is a lot of data to manage. Our data is uniquely characterized by how much is missing. It may be more efficient to not eagerly infer and store the missing data, but instead store the inference procedures, and fill in the missing data at run-time. Finally, our data comes with not only probabilistic uncertainty, but also uncertainty bounded by hard constraints.

## 6 Role of Models

In Section 1 we advertise the data-driven approach as an alternative or complement to model-driven, where we do not produce query results generated from models in lieu of the actual measured data. We may appear to then contradict ourselves by proceeding to discuss models extensively in every section and how they can be used to make data collection more efficient. We now provide discussion to reconcile this apparent conflict.

The key distinction between data-driven and model-driven is the degree of trust placed on models. In model-driven, models can serve as substitutes for the readings generated in the network. In data-driven, models are used for optimization, but do not affect correctness. We design suppression schemes based on models of network behavior. If such models turn out to be inaccurate or imprecise, correctness of the collection is not affected; only efficiency is. If we know nothing about a deployment or the nature of the measurements we are taking, we may initially pay a high cost for collection. Eventually, however, we will collect enough data under a large variety of conditions from which to build models. These models are worth the effort, and more valuable than if handed over by a domain expert; they will be exactly specific to the deployment and its node locations. A positive feedback loop develops: the more data collection we perform, the more we learn about the network, and the better we can optimize subsequent collection.

One of the important problems in data-driven acquisition is automating the refinement process for users not interested in constructing models. This allows the vineyard owner who has purchased a ready-to-deploy sensornet kit [5] to remain oblivious to the use and refinement of models, except to notice the improvement in energy efficiency that lets him replace batteries less often.

**Reliance on Models** Data-driven is not a replacement for model-driven. Instead, it fills in a trade-off spectrum between trust of models and energy cost. The less a user is willing to allow models to influence query results, the more they must pay for actual checks against the data. In this paper the trade-off appears most prominently in Section 3, when discussing failure. Because we cannot fully eliminate failure, it is extremely expensive, if not impossible, to meet the extreme where models have no effect on correctness. The inference process that produces query results combines data-driven and model-driven concepts. Data-driven includes the suppression scheme parameters (i.e., what a non-report, assuming no failure, means) and redundancy that contribute constraints on the possible reconstructions. Model-driven includes the use of a model that probabilistically biases reconstructions. The influence of each on these approaches is tunable. It is possible to use a very uninformative model that in conjunction with high failure and/or little redundancy, will result in uninformative query results. In this case we get a poor query result, but the model has little influence. It is possible to add more redundancy such that, even with the uninformative model, the possible reconstructions are constrained enough to make inference informative. In this case we get a good query result without much influence by the model, but have pay the added cost for redundancy, either by monitoring more suppression links, or transmitting longer reports for each.

We expect users to tune the trade-off between data-driven and

model-driven as their deployment matures. Initially, when nothing is known about the deployment, it is worth the extra energy to not be biased, and perhaps misled, by informative prior models. Once accurate models of the deployment are constructed, however, it may be acceptable to rely on those models and save energy in collection. Still, when unusual conditions emerge naturally, or artificially for experimental purposes, the existing model may no longer hold, and it is best to temporarily shift back toward data-driven.

## 7 Related Work

**Model-Driven vs. Data-Driven** Sensornets typically exhibit correlation. The model-driven approach is the first to aggressively exploit this by constructing a model over all nodes, times, and sensor types [10, 9]. If information gleaned from readings from some arbitrary point in the network somehow predicts readings in some other arbitrary point, or at the same point in the future, these correlations can be encoded. In parallel, we see a number of data-driven approaches exploiting spatial correlation in local ways, typically through clustering [14, 21], and exploiting temporal correlation on a node-by-node basis [17, 6]. The former case focuses on having a representative set of nodes stand in for all others, often with the requirement each node be within one hop of its representative, for ease of checking if each is accurately represented. The latter case focuses on fitting stretches of data from individual nodes to models, such that parameters can be transmitted in place of most values. More recently, model-driven researchers have investigated maintaining models in-network, rather than out [2, 25].

Model-driven faces the problem of presenting query results, given their inherent uncertainty, to users, especially those without statistics backgrounds [9]. We face the same problem in data-driven, especially due to the influence of message failure. The avenue of presenting a graphical interface to depict possible value settings is promising. For users (or automated systems) that only care about reaching particular certainty thresholds, we favor the post-processing described in Section 3, combined with future tuning of the suppression scheme (including redundancy).

**Queries** Ours is not the first paper to suggest `SELECT *` is a key query and of special importance to collaborating scientists. This is evidenced by the sheer volume of work focused on total data collection [17, 6, 25, 2, 23]. This list overlaps strongly with our above list of prior data-driven research. There has also been considerable work on specific continuous queries, including Deligiannakis *et al.* on continuous aggregates using distributed error budgeting [7] and our own work on extreme value queries [24].

**Failure** The vulnerability of sensornets to message loss has been examined at all network layers. One major finding is edges between network nodes do not either exist or not exist [26, 27]. Rather, links exhibit a range of reliabilities. At the communication layer, it is important to test edges and prioritize with discretion which can be used for routing. Note this step returns physical link statistics, different than the  $n^2$  virtual links we use as input to milestone optimization in Section 4.1.

The lower network layers can minimize the probability messages are not received by having recipients send acknowledgments back to the sender, and having the sender re-transmit if no acknowledgment is received. The number of attempts is capped. The application layer provides the opportunity to make data collection robust to failure; that is, arbitrary messages can fail without greatly compromising the application's result. One interesting example of this occurs in in-network aggregation. For duplicate-insensitive aggregates like `MAX`, nodes can transmit multiple copies of their values toward the root, assuming all will not fail, without affecting the

result. This strategy fails for duplicate-sensitive aggregates like `COUNT` and `MEAN`. The sketch [4] aggregates by hashing each node id and value pair, and setting a subset of its bits accordingly. Additional copies of the same pair will not affect the sketch's setting. At the root the overall setting of bits is used to approximate the aggregate value.

The problem of missing data, the resulting ambiguity, and what to infer about the true values, arises in a number of contexts. Khoussainova *et al.* describe such a problem for RFID sensors [19]. If a particular tag is not scanned by a reader, is it not in the reader's area, or is it present, but failed to be read? Their solution, similar to ours in Section 3, is to add integrity constraints into the reconstruction process (e.g. if a tag was detected by a reader in the previous timestep but not by any in the current, it is more likely to be near that reader than any other). The main difference from our approach is these constraints are chosen offline by the user, whereas our constraints, encoded in suppression links, are actually monitored in the network. These constraints then influence inference.

**Application/Communication Layer Interaction** Our work on layer interaction is motivated by the need to structure the boundary between them. Not surprisingly, there have been a number of efforts focused on this boundary. For applications that transmit all data to the root, Pattem *et al.* differentiate compression-driven routing and routing-driven compression [22]. When the network exhibits strong spatial correlations among nodes, it is important such values are routed to common points as quickly as possible using compression-driven routing. Here the data is compressed, to be sent on the root at reduced bandwidth. When there is little spatial correlation, routing-driven compression is employed. Values are transmitted to the root along the most efficient routes, and any compression is done opportunistically.

Hellerstein and Wang pose the question of whether very specific applications can be mapped onto the communication layer [15]. They focus on the Haar wavelet compression structure, where the routing tree topology must match a rigidly structured Haar support tree. Regardless of what single hop links are actually viable in the network, using virtual edges (Section 4) overlaid on the network, it is certainly possible to achieve this goal. The real question is what the message costs are for supporting the Haar wavelet. Milestone optimization is a framework for answering this. The choice of virtual edges is constrained because only particular subsets constitute a wavelet support tree. For each such subset, the bandwidth over a virtual edge depends on the values converging at it.

## 8 Conclusion

We have introduced the data-driven approach for collection in sensornets. Our motivation is to support continuous queries without continuous data streams, while using the data generated in the network as the ground truth. The primary technique for achieving this is suppression, combined with use of models. We use models to optimize acquisition, but not at the cost of correctness. Ultimately, due to failure, it is difficult to completely eliminate reliance on models. We discuss techniques for coping with failure by adding redundancy. We also discuss the milestone framework for managing interaction between, but not merging, the application and communication layers.

In this paper we have not declared how data acquisition should be performed, but instead have provided general techniques that expose trade-offs so application designers can make informed decisions. Our suppression scheme definition generalizes a whole class of applications and allows for consistent and simple comparison among these. There is a trade-off between allowing models to

stand in for data and energy cost, particularly in handling failure. We leave it to designers to choose points along this trade-off, but ensure the consequences of these decisions are exposed, in terms of energy cost and certainty of results. The milestone overlay addresses the problem of building applications while accounting for routing costs, without needing to know details about the routing protocol itself.

We anticipate a tremendous amount of work on application development and are eager to see how the techniques in this paper aid in that process and/or can be used to characterize the results for the sake of comparison. We plan to develop more sophisticated suppression schemes that delve deeper into encoding spatial constraints. As we design such schemes, we will also address the failure problem to find new ways to add redundancy and incorporate it into inference. We also plan to develop sample applications using milestone and determine if its interface gleans enough information from the communication layer to accurately predict cost, so it can then be used for application design.

## References

- [1] TinyOS. [www.tinyos.net](http://www.tinyos.net).
- [2] D. Chu, A. Deshpande, J. Hellerstein, and W. Hong. Approximate Data Collection in Sensor Networks using Probabilistic Models. In *ICDE*, Atlanta, Georgia, USA, Apr. 2006.
- [3] D. Chu, K. Lin, A. Linares, G. Nguyen, and J. Hellerstein. sllip: A Sensor Network Data and Communications Library for Rapid and Robust Application Development. In *IPSN*, Nashville, Tennessee, USA, Apr. 2006.
- [4] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate Aggregation Techniques for Sensor Databases. In *ICDE*, Boston, Massachusetts, USA, Mar. 2004.
- [5] Crossbow Inc. *Smart Dust Application Note*.
- [6] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Compressing Historical Information in Sensor Networks. In *SIGMOD*, Paris, France, June 2004.
- [7] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Hierarchical In-Network Data Aggregation with Quality Guarantees. In *Intl. Conf. on Extending Database Technology*, Heraklion, Crete, Mar. 2004.
- [8] A. Deshpande, C. Guestrin, W. Hong, and S. Madden. Exploiting Correlated Attributes in Acquisitional Query Processing. In *ICDE*, Tokyo, Japan, Apr. 2005.
- [9] A. Deshpande, C. Guestrin, and S. Madden. Using Probabilistic Models for Data Management in Acquisitional Environments. In *CIDR*, Asilomar, California, USA, Jan. 2005.
- [10] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-Driven Data Acquisition in Sensor Networks. In *VLDB*, Toronto, Canada, Aug. 2004.
- [11] A. Deshpande and S. Madden. MauveDB: Supporting Model-based User Views in Database Systems. In *SIGMOD*, Chicago, Illinois, USA, June 2006.
- [12] R. Fonseca, S. Ratnasamy, J. Zhao, C. T. Ee, D. Culler, S. Shenker, and I. Stoica. Beacon-Vector Routing: Scalable Point-to-Point Routing in Wireless Sensor Networks. In *Proc. of the 2005 Symp. on Networked Systems Design and Implementation*, Boston, Massachusetts, USA, May 2005.
- [13] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden. Distributed Regression: an Efficient Framework for Modeling Sensor Network Data. In *Proc. of the 2004 IPSN*, Berkeley, California, USA, Apr. 2004.
- [14] H. Gupta, V. Navda, S. Das, and V. Chowdhary. Energy-Efficient Gathering of Correlated Data in Sensor Networks. In *ACM Intl. Symp. on Mobile Ad Hoc Networking and Computing*, Urbana-Champaign, Illinois, USA, May 2005.
- [15] J. Hellerstein and W. Wang. Optimization of in-network data reduction. In *Proc. of the 2004*, Toronto, Canada, Aug. 2004.
- [16] B. Hull, K. Jamieson, and H. Balakrishnan. Mitigating Congestion in Wireless Sensor Networks. In *SENSYS*, Baltimore, Maryland, USA, Nov. 2004.
- [17] A. Jain, E. Chang, and Y. Wang. Adaptive Stream Resource Management Using Kalman Filters. In *SIGMOD*, Paris, France, June 2004.
- [18] S. Jeffery, G. Alonso, M. Franklin, W. Hong, and J. Widom. Declarative Support for Sensor Data Cleaning. In *Intl. Conference on Pervasive Computing*, Dublin, Ireland, May 2006.
- [19] N. Khoussainova, M. Balazinska, and D. Suci. Towards correcting input data errors probabilistically using integrity constraints. In *Proc. of the 2006 ACM Workshop on Data Engineering for Wireless and Mobile Access*, Chicago, Illinois, USA, June 2006.
- [20] Y. Kim, R. Govindan, B. Karp, and S. Shenker. Geographic Routing Made Practical. In *Proc. of the 2005 Symp. on Networked Systems Design and Implementation*, Boston, Massachusetts, USA, May 2005.
- [21] Y. Kotidis. Snapshot Queries: Towards Data-Centric Sensor Networks. In *ICDE*, Tokyo, Japan, Apr. 2005.
- [22] S. Patten, B. Krishnamachari, and R. Govindan. The Impact of Spatial Correlation on Routing with Compression in Wireless Sensor Networks. In *IPSN*, Berkeley, California, USA, Apr. 2004.
- [23] A. Silberstein, R. Braynard, and J. Yang. Constraint-Chaining: On Energy-Efficient Continuous Monitoring in Sensor Networks. In *SIGMOD*, Chicago, Illinois, USA, June 2006.
- [24] A. Silberstein, K. Munagala, and J. Yang. Energy-Efficient Monitoring of Extreme Values in Sensor Networks. In *SIGMOD*, Chicago, Illinois, USA, June 2006.
- [25] D. Tulone and S. Madden. PAQ: Time Series Forecasting for Approximate Query Answering in Sensor Networks. In *EWSN*, Zurich, Switzerland, Feb. 2006.
- [26] A. Woo, T. Tong, and D. Culler. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In *Proc. of the 2003 SENSYS*, Los Angeles, California, USA, Nov. 2003.
- [27] J. Zhao and R. Govindan. Understanding Packet Delivery Performance in Dense Wireless Sensor Networks. In *Proc. of the 2003 SENSYS*, Los Angeles, California, USA, Nov. 2003.