Constraint Chaining: On Energy-Efficient Continuous Monitoring in Sensor Networks*

Adam Silberstein Rebecca Braynard Jun Yang
Department of Computer Science, Duke University, Durham, NC, 27708, USA
{adam,rebecca,junyang}@cs.duke.edu

ABSTRACT

Wireless sensor networks have created new opportunities for data collection in a variety of scenarios, such as environmental and industrial, where we expect data to be temporally and spatially correlated. Researchers may want to continuously collect all sensor data from the network for later analysis. Suppression, both temporal and spatial, provides opportunities for reducing the energy cost of sensor data collection. We demonstrate how both types can be combined for maximal benefit. We frame the problem as one of monitoring node and edge constraints. A monitored node triggers a report if its value changes. A monitored edge triggers a report if the difference between its nodes' values changes. The set of reports collected at the base station is used to derive all node values. We fully exploit the potential of this global inference in our algorithm, CONCH, short for *constraint chaining*. Constraint chaining builds a network of constraints that are maintained locally, but allow a global view of values to be maintained with minimal cost. Network failure complicates the use of suppression, since either causes an absence of reports. We add enhancements to CONCH to build in redundant constraints and provide a method to interpret the resulting reports in case of uncertainty. Using simulation we experimentally evaluate CONCH's effectiveness against competing schemes in a number of interesting scenarios.

1 Introduction

Wireless sensor networks have the potential to enable data collection on an unprecedented scale. They have a range of scientific, industrial, and military applications. For example, our collaborators in environmental science have deployed a sensor network in a forest to collect light, temperature, soil moisture, and sap flow data to understand how environmental changes influence the growth, survival, and reproduction of trees. The key challenge in this project is to collect data from the network efficiently and continuously for analysis. A straightforward approach is to instruct all nodes in the sensor network to send their readings at regular intervals to a base station. Periodic reporting, however, is at odds with one of the major concerns for wireless sensor networks: energy. Nodes have limited battery life, and radio transmission is the primary consumer of energy. Excessive transmission quickly depletes batteries, rendering the nodes useless.

If we can describe the data to be collected as a declarative query, we can "push" the query down into the sensor network—an approach pioneered by systems such as Cougar and TinyDB [19, 10]. Given the query, we can perform filtering and aggregation in the network to reduce the amount of data transmitted to the base station. For example, if we only need to obtain the highest temperature reading from the network, only one reading need reach the base station; other readings can be dropped on their way to the base station as soon as they encounter another reading that is higher. In many situations, however, users may want to collect *all* sensor readings, without any filtering or aggregation, to produce a dataset that will support offline, ad hoc analysis later. This situation is especially common in scientific applications of sensor networks, where analysis of data is often exploratory in nature. For example, our collaborators in environmental science would like to collect all temperature readings (up to a certain precision) in an area over a time, which can then be used offline to develop and evaluate sophisticated statistical models of tree growth.

There are several opportunities for reducing the cost of monitoring and reporting a group of sensor readings, independent of query type. First, readings often change slowly over time and do not usually deviate from their expected values. For example, in industrial applications, a change in some monitored quantity may be a warning of impending failure and occurs rarely. Second, readings are often spatially correlated. In habitat monitoring, if one sound sensor detects a loud cry from an animal, its neighboring sensors likely hear the same sound. In general, many types of sensor data exhibit strong correlation in both space and time. For example, if one sensor detects high soil moisture due to precipitation, sensors in neighboring areas with similar soil composition and elevation likely observe similar moisture readings; furthermore, soil moisture usually stays at a saturated level during precipitation, and tapers off to a normal level afterward. A primary goal of this paper is to investigate how to make continuous monitoring in sensor networks more energy-efficient by exploiting these opportunities.

We use the term *suppression* to refer generally to query-independent techniques for reducing the cost of reporting changes in sensor values. We outline several basic techniques below, and motivate the need for developing better ones.

Temporal Suppression In this scheme, a node does not transmit a value if it has not changed since last reported. The base station, in turn, assumes any unreported values remain unchanged. This scheme is effective when values seldom change. On the other hand, when large-scale change occurs in an area, all nodes must report to the base station, incurring a high cost.

*The authors are supported by NSF CAREER award IIS-0238386, NSF grant CNS-0540347, and an IBM Faculty Award.

SIGMOD 2006, June 27–29, 2006, Chicago, Illinois, USA.

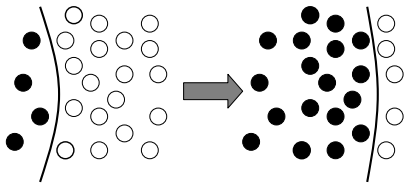

Figure 1: Moving front of a phenomenon.

Spatial Suppression In the basic version of this scheme, a node suppresses its value if it is identical to those of its neighboring nodes. Meng et al. [12] suggests a more effective scheme based on averages, in which all nodes attempt to report their values at different time slots during a logical timestep. Nodes overhear reports from their neighbors. When a node's slot comes up, it first computes the average of all values overheard so far. If its value equals this average, its report is suppressed. The base station later fills in missing values with the average of the neighboring values. This approach does introduce some complications. To accurately derive a node's value, the base station must know which neighbors were averaged when suppression was triggered; the average of this subset may be different from the average of all neighbors. Resolving this discrepancy requires a global reporting order of all nodes, maintained by the base station. Holding to a global order in the presence of node clock skew is likely expensive for sensors. We ignore this overhead, however, for the sake of comparison with more advanced schemes.

Spatio-Temporal Suppression The next step is to combine the advantages of both spatial and temporal suppression techniques. One idea is to suppress nodes if they qualify for either type of suppression. This idea is flawed, however, because it is impossible for the base station to correctly derive unreported values. The "or"ing of the schemes creates ambiguity. The base station does not know if a missing value is temporally or spatially suppressed; if the values derived by assuming each case turn out to be different, there is no way to determine the correct value. Another possibility is to "and" the schemes, suppressing only if a node qualifies for both. This approach, however, would be less effective than using only one type of suppression. Fewer nodes can be suppressed both temporally and spatially than can be suppressed one way or the other. Therefore, "and"ing the schemes would only hurt efficiency.

Nevertheless, there is potential in combined suppression. If values do not change, they should not be reported. In addition, if they do change, but the relationship between neighboring nodes remain the same, we may suppress some reports. Consider the network fragment shown in Figure 1. In one timestep, a physical phenomenon causes a band of nodes to all rise in value (shift from white to black). Discretized environmental readings, e.g., temperature or moisture, often exhibit this type of correlated behavior, where values in the interior of the band change, but maintain the same relative relationships with their neighbors. We want to suppress these changes. Ideally, only one node on each boundary (old and new) of the band needs to report the change that is consistent within the band. In contrast, with pure temporal suppression, all nodes in the band must report. With pure spatial suppression, all nodes on the band boundary must report; for every local cluster of nodes within the band, at least one node must report.

Our Contributions As we have illustrated, it is not obvious how to design an effective spatio-temporal suppression policy. Our primary goal, therefore, is to develop a monitoring algorithm that fully exploits the potential of spatio-temporal suppression in minimizing the energy cost. An additional challenge we address is handling of failures, which can have particularly bad effects on data quality of highly efficient suppression techniques. If the base station does not hear from a node, there may be no change to report, but it could be the case that an update message was lost. In the worst case, a single lost update message can affect the data quality in a large area; for example, with ideal spatio-temporal suppression for the scenario in Figure 1, if the update message from the boundary node is lost in transmission, the base station will miss the new values in the entire band. We investigate how to augment our monitoring algorithm in the presence of failures, which are common in wireless sensor networks deployments. Specifically, our contributions in this paper include the following:

- We present a spatio-temporal suppression technique called CONCH, short for *constraint chaining*. By monitoring a combination of individual values (as *node constraints*) and relationships between neighboring values (as *edge constraints*), CONCH exploits spatio-temporal correlations in data much more effectively than previous suppression schemes. CONCH "chains" together locally monitored node and edge constraints into a network of constraints spanning all nodes. This constraint network allows us to use a change detected *locally* at one point in the network to *globally* infer values at nodes around the network, without incurring the cost of communicating between these nodes. For example, with these features, CONCH is able to achieve ideal spatio-temporal suppression for the scenario in Figure 1.
- We develop cost-based optimization techniques for constructing CONCH monitoring plans aimed at minimizing the total energy consumption in the sensor network.
- To cope with failures in the sensor network, we propose to augment a minimum-cost CONCH monitoring plan with redundancy. We present a framework for removing data inconsistency caused by failures and for recovering correct values. We provide a computationally feasible method for recovering the most likely correct values under simplified value and failure models.
- We experimentally evaluate CONCH against other monitoring algorithms using simulation, and demonstrate its advantages (often an order of magnitude reduction in energy costs) for a number of representative scenarios. Our experiments also show the effectiveness of our failure-handling techniques.

2 Related Work

Suppression A number of papers support monitoring queries using suppression. One approach that closely compares to ours is *event contour* [12]. This method uses temporal suppression to only report values when they change by a significant amount, and applies a neighborhood approach to invoke spatial suppression as well. When a node attempts to report its value to its parent node, it must compete with neighboring nodes that also wish to send. While it waits for a chance, the node overhears the values transmitted by its neighbors. If these values average to within some threshold of the node's, it suppresses its value. Therefore, if a neighborhood contains nodes all with similar values, not all will be sent. Nevertheless, we are left with a number of problems. First, we can never suppress all readings in an area, even if all are the same. Second, if a pair of nodes have different values, but are highly correlated such that they always move together, we will always report both, even though one is sufficient to derive the other. Finally, the spatial component suffers from the flaw mentioned in Section 1 where the base station does not know on which values suppression is based.

Solis and Obrazcka [17] present a solution for continuously maintaining isoline maps. They recognize the need to combine spatial and temporal suppression. Their algorithm is very similar to our

precursor NEIGHBORHOOD scheme presented in Section 3. Unlike our scheme, though, they report whenever a difference between neighbors is detected; they miss out on the chance to suppress values that differ, but whose relative difference remains consistent. Both suffer from redundant reporting, as we describe in Section 3.1. CONCH resolves this issue.

Chintalapudi and Govindan [1] address the problem of edge detection and returning enough data so the root can construct an accurate depiction of the boundaries of some phenomenon. Nodes can individually determine if they reside within the phenomenon, but must consult other nodes within some radius to determine if they are on the edge of it (i.e., if there is a neighbor not registering the phenomenon). If nodes contact too large a neighborhood, the scheme risks sending redundant data to the root. If nodes contact too small a neighborhood, the scheme risks missing some data and mis-setting the boundary.

Temporal suppression fits naturally with continuous queries. One example, [16], temporally suppresses nodes from reporting their values if they have not changed by more than some percent since previously reported. This bounded approximation supports continuous aggregate queries with bounded error. Less is at stake in suppression for aggregates, whose semantics naturally imply compression, than monitoring all values, where as many as all values are potentially returned.

Spatial Suppression with Representative Nodes A managed approach to avoid sending correlated data to the root is to gather such data at intermediate points [13, 14]. The main observation is if a set of data is highly correlated, rather than sending values to the root by the shortest route, it is beneficial to send them to a mutual gathering point. At this point the correlation is discovered, and only non-redundant values are sent on to the root. This technique is essentially one of spatial suppression. Because these approaches are tailored to ad hoc queries, neither considers temporal suppression. In [13] nodes are organized into clusters; all nodes in a cluster send their values to a cluster head node; suppression happens at the cluster head and also en route. The other approach, [14], declares regions of interest and arranges for nodes within a region to send to a common border node, where suppression can occur. In both cases, opportunities to suppress across clusters/regions are not exploited. Furthermore, the cluster shapes are fixed based on network topology and are not tailored to actual correlation patterns. The regions of interest are set by the root and the cost for adjusting the regions is significant. The rigidity of these may lead to inefficiency. These are issues we solve with CONCH, whose planning exploits correlation patterns, and can adapt to changes.

Chu et al. propose the *Ken* [2] framework for suppression in continuous monitoring. It uses a joint probability model to suppress as much data as possible from reaching the root, while still allowing the root to derive suppressed values from those reported with some level of certainty. They propose a disjoint-clique approach to divide nodes into groups and then build models for each of these. Temporal suppression cannot be fully achieved since all values are sent to clique roots each round. There is a trade-off between temporal and spatial suppression. The larger the cliques, the more spatial can be exploited, but the further values must be sent each round.

Another category of work chooses representative nodes from neighborhoods to exploit spatially correlated data. Examples are *snapshot queries* [9] and *connected k-coverage* problem [20]. These choose a subset of nodes to respond to queries, where each chosen node stands in for neighboring nodes within some error. This work leverages correlation as we do, and implies spatial suppression when they allow some nodes to ignore queries. The use of coverage nodes means results are approximate, although with some error guarantee. Therefore, regular checks must be done between nodes and their representatives. This step is not discussed in [20]. In the case of [9], these look similar to our "update" messages, though theirs are sent with regular frequency, while ours are triggered by value change. The use of coverage nodes ensures some number of nodes respond to every query. Both approaches require every node be represented by a node within communication distance. Therefore, this approach can never achieve our goal of near or total suppression. Finally, these schemes are not tailored to continuous queries, so do not address temporal suppression.

A common theme among the preceding approaches is that certain nodes become responsible for reporting for their regions and must report in the event of change. CONCH, as we will show, differs from these because it uses constraint chaining. It is possible for a region to experience change but have no node from that region report to the root. Such changes can be inferred using change reports from distant nodes, or from changes at the root itself.

Model-Based Suppression We have already seen the use of models for spatial suppression in [2]. An example of temporal suppression using models rather than values is [8]. They use Kalman Filters to suppress node data as long as it fits the current model. This work is orthogonal to ours, and exactly the type of sophisticated modeling we think can be plugged into CONCH to direct suppression. A strategy in [6] is to buffer large amounts of measurements at each node. Rather than transmit the buffer contents, they identify and transmit a *base signal* of a few parameters, which can be used to estimate all measurements. This approach is again orthogonal to ours; instead of monitoring actual values, we can apply our techniques to monitor the parameters of the base signal.

Failure The more effectively suppression is exploited, the more damage can be done by misinterpreting a failed message as a suppressed message. Failure handling has been approached by use of redundancy in the context of routing and ad hoc querying. For example, [11] divides routing into graph and tree portions, where data is initially sent up multiple paths to protect against failure. The higher the risk of failure, the larger the graph portion. Failure is not as well studied in association with suppression. *TiNA* [16] uses heartbeat message to check if nodes that have been suppressing for some time are still alive. Unless these are sent with high frequency, however, there is a risk that failure will go undetected for a long time and affect many rounds of results. On the other hand, sending them with high frequency defeats the purpose of suppression.

3 Preliminaries

Sensor Network A sensor network consists of a set N of n fixed-location nodes $\{u_i \mid 1 \leq i \leq n\}$, each measuring a value v_i. One node serving as the base station is a full-scale computer with no energy limitations. The base station knows all nodes and their locations, and is responsible for planning queries and monitoring tasks and extracting and reconstructing values from the network. A communication edge e_{ij} exists between any pair of nodes u_i and u_j within communication distance. A communication network utilizing these edges connects all nodes to the base station, either directly or through other nodes. Any existing protocol [7, 18] can be used for routing.

The primary use of energy in sensor nodes is for radio communication. The cost of transmitting data dwarfs the cost of doing computation [15]. Therefore, we optimize energy efficiency by minimizing the number and size of messages sent through the network, and evaluate all algorithms on this metric. The total amount of energy spent in sending a message with x bytes of content is given by $\sigma_s + \delta_s x$, where σ_s and δ_s represent the per-message and per-

byte sending costs, respectively. As an example, typical values for MICA2 motes [5] are $\sigma_s = .645$mJ and $\delta_s = .0144$mJ/byte. Receiving cost is defined analogously, with typical values of σ_r and δ_r roughly 60% less than their sending counterparts.

For a continuous query or monitoring task, the base station initially disseminates a query or monitoring plan, consisting of instructions to be carried out at runtime by individual nodes, into the network. Since this dissemination occurs only once and its cost is amortized over the lifetime of the task, we ignore it in cost-based optimization. We also permit the plan to be updated at runtime, but infrequently so that it has overall negligible effect on the amortized energy consumption.

Problem Statement In each timestep, each node u_i produces a new value for v_i. Our goal is to collect, at the base station, all values produced in every timestep. For simplicity of discussion, we assume that these values are discretized according to the precision requirement of the data collection task. For example, if we wish to collect temperature at precision of 1 degree Celsius, each temperature reading can be discretized to an integer, providing an error bound of ± 0.5 degree Celsius.

Note that our "values" are much more general than just the sensor readings; they can be quantities derived from readings or even a history of readings through model fitting. This generalization makes our techniques applicable in conjunction with sophisticated model-based compression schemes such as [8, 2]. As a very simple example, suppose a temperature reading x can be predicted by a function $f(t, p)$ with reasonable accuracy, where t is the time of the day and p is a model parameter that intuitively captures the "baseline" temperature for the day independent of t. Instead of tracking x as a "value" in our approach, we can track two "values": p and the quantity $e = x - f(t, p)$ (predication error). Model parameter p changes much less than x, and if the model is accurate, the e should remain 0 most of the time (with appropriate discretization). Hence, monitoring p and e should be much cheaper than monitoring x, which can be reconstructed as $f(t, p) + e$. Our techniques then help with monitoring of p and e, by further exploiting the fact that they do not change often, and their values at neighboring nodes are spatially correlated —p's are naturally correlated in space, while some neighboring e's may change together due to movement of canopy shades.

3.1 A First Cut: NEIGHBORHOOD

Before delving into CONCH, we first present an algorithm called NEIGHBORHOOD that serves two purposes. First, it introduces an idea for combining spatial and temporal suppression that will also be used by CONCH. Second, by analyzing its shortcomings, we gain insights that lead us to the development of CONCH. We call the set of nodes within communication distance of u_i the *neighborhood* of u_i, denoted N_i. Note that $u_i \in N_j$ implies $u_j \in N_i$, assuming communication is always bidirectional. The main idea of NEIGHBORHOOD is for each node u_i to maintain d_{ij}, the difference in value between itself and each node $u_j \in N_i$.

In every timestep, each u_i broadcasts its value v_i^{new} to all nodes in N_i only if its value has changed since the value v_i^{old} from the previous timestep—as in temporal suppression. Then, u_i updates d_{ij} for each neighbor u_j as follows. If u_i receives a broadcast message from u_j with updated value v_j^{new}, u_i updates d_{ij} to $v_i^{new} - v_j^{new}$; if u_i has not received any broadcast message from u_j by the end of the timestep, u_i infers that v_j remains unchanged, and updates d_{ij} to $d_{ij} - v_i^{old} + v_i^{new}$. At the end of the timestep, for each d_{ij} that has been updated to a different value in this timestep, u_i sends d_{ij} to the base station if $i > j$. As an optimization, these updates can be grouped into one message. In essence, d_{ij} is a spatial relationship

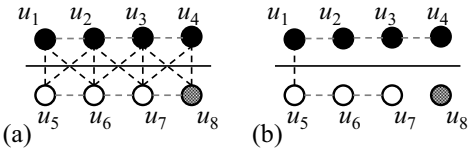

Figure 2: Edge monitoring in NEIGHBORHOOD (a) vs. node/edge monitoring in CONCH (b).

suppressed temporally.

In NEIGHBORHOOD, the base station knows the difference (d_{ij}) in value across every communication edge at all times. Thus, NEIGHBORHOOD is an *edge monitoring* scheme. Also, the value at the base station is always known (if the base station does not monitor any value, it simply monitors a "dummy" value of zero). From this value, it is easy to reconstruct all values in the network in one pass with a traversal of the network topology. During this traversal, suppose we have already reconstructed v_i; then for each neighbor u_j of u_i, we can reconstruct v_j as $v_i - d_{ij}$. Note this traversal is done completely at the base station with its knowledge of the network topology, without any communication to the real network.

NEIGHBORHOOD improves over the suppression schemes discussed in Section 1. Once again, suppose two neighbors share the same value; if both values remain constant or rise in tandem, we can suppress reporting of their difference to the base station. Moreover, even if two neighbors have different values that both change, as long as the difference in value remains constant across changes, we can suppress reporting.

Shortcomings While NEIGHBORHOOD combines spatial and temporal suppression effectively, it still has important flaws. These are best seen with examples. Say we start with a collection of 8 nodes all with the same value, v_l, and in the next timestep, the 4 nodes above jump to a higher value, v_h, as depicted in Figure 2(a). A horizontal axis shows the split in values. The dotted lines indicate neighbor relationships between pairs of nodes. When the 4 nodes above rise in value, the value differences along all edges across the split (colored in black) change. For each such edge, NEIGHBORHOOD must update the base station. Obviously, there are more updates than necessary. For instance, the base station should be able to infer the new value at node u_1 from just one monitored edge, say e_{15}, but the other edge, e_{16}, is also reported. This problem is inherent in NEIGHBORHOOD because it monitors all communication edges, most of which are redundant; for example, d_{13} should always be the same as $d_{12} + d_{23}$. This redundancy causes lots of unnecessary update traffic, a problem that we shall address in CONCH with more judicious choice of edges to monitor. Although there are lots of value changes in the network in this example, the overall effect can be described succinctly: all nodes above the axis move to v_h. With CONCH, we will be able to use one short message to capture this overall effect.

Another shortcoming of NEIGHBORHOOD arises in its handling of "outlier" nodes whose values simply change haphazardly without any spatio-temporal correlation. Suppose u_8 turns out to be such a node in Figure 2(a). NEIGHBORHOOD monitors all edges incident to u_8, but v_8 changes haphazardly, causing updates to all these edges in every timestep. CONCH will need to be able to identify such outliers and protect the rest of the monitoring plan from being affected by them.

4 Min-Cost CONCH

We now present CONCH, the main contribution of this paper. The crux of CONCH is that we can provide effective spatio-temporal suppression using a minimum spanning forest covering all nodes in

the sensor network; the roots and edges of this forest correspond to monitored constraints. Like NEIGHBORHOOD, CONCH uses edge monitoring to exploit spatio-temporal correlations in data, and relies on "chaining" of value differences along monitored edges to recover node values. The key observation here is, to be able to reconstruct all values in the network, we simply need to monitor all edges in a spanning tree, covering all nodes. For the purpose of reconstructing node values, any spanning tree suffices, but we should choose one that is the cheapest to monitor; intuitively, we prefer those edges connecting values that tend to change in tandem. A further improvement over NEIGHBORHOOD is that CONCH allows nodes to be directly monitored (using just temporal suppression). By relaxing the requirement that every value must be monitored in conjunction with a neighbor, CONCH makes it easier to separate uncorrelated areas of values and, in particular, to isolate outliers (as discussed at the end of Section 3.1). This flexibility implies a spanning forest of trees, whose roots are directly monitored.

As an example of how CONCH works, consider the same scenario as in Section 3.1. Figure 2(b) illustrates how CONCH might choose to monitor this network. The spanning forest consists of one skinny tree (in fact just a single path) that connects the top 4 nodes with 3 left-most bottom ones, plus one singleton-tree with just u_8. The first thing to notice versus NEIGHBORHOOD is that far fewer edges are monitored. The second is that when the top nodes change to v_h, only one edge (e_{15}) observes a change in value difference, so only one update message is sent to the base station. Because none of the edges between black nodes were reported, we know all of those edges must still have the same value; if u_1 now has value v_h, then so do u_2, u_3, and u_4. Finally, note CONCH directly monitors the outlier, u_8, and does not monitor any edges incident to u_8, because there is no benefit, and only overhead cost, in monitoring u_8 in conjunction with an uncorrelated value.

This rest of this section starts with a formal description of a CONCH plan and discusses how the plan is carried out at runtime to perform lossless data collection, assuming no failure, from the sensor network (Section 5 addresses failure). We then discuss how to obtain a min-cost CONCH plan, i.e., one that is expected to spend the least amount of energy over time.

4.1 CONCH Plan

Suppose the sensor network consists of a set of nodes N and a set of communication edges E. Formally, a CONCH plan is specified by a set of nodes $N_m \subseteq N$ (called the *monitored nodes*), a set of (undirected) edges $E_m \subseteq E$ (called the *monitored edges*), and a *reporter assignment* function $rep : E_m \rightarrow N$ that maps each monitored edge $e_{ij} \in E_m$ to one of u_i and u_j. We call $rep(e)$ the *reporter* of e, and the other node incident to e the *updater* of e. Nothing prevents a node from serving either role on behalf of any of its incident edges, but each node serves only one role per edge.

A *valid* CONCH plan satisfies the following requirements: (1) the base station is in N_m; (2) every node $u \in N$ is either in N_m (directly monitored), or there exists a path from some node in N_m to u (indirectly monitored), where every edge on this path is in E_m. This section focuses on *minimal* CONCH plans, valid plans for which it is impossible to remove some node from N_m or some edge from E_m while still maintaining the validity of the plan. It is easy to see that we can construct a minimal CONCH plan from a spanning forest of the sensor network.

The potential savings of a minimal CONCH plan over NEIGHBORHOOD can be seen in part by calculating the number of quantities monitored. A minimal CONCH plan, by definition, contains no more than n monitored quantities. In contrast, the number of edges monitored by NEIGHBORHOOD is $n/2$ multiplied by the average number of neighbors per node, which is likely much higher. The potential savings comes from the flexibility of requiring far fewer monitored quantities and then choosing them carefully.

Intuitively, CONCH monitors a network of constraints. Each node in N_m maintains a node constraint, reporting to the base station whenever its value changes. Each edge in E_m is an edge constraint, jointly maintained by a reporter and an updater: the updater notifies the reporter whenever the updater's value changes, and the reporter notifies the base station whenever the value difference along the edge changes. The chaining of node and edge constraints allows the base station to recover all values in the network. We next describe the operation of CONCH in detail.

Start-Up Phase Once a CONCH plan has been constructed at the base station, we disseminate it into the network. Each node $u_i \in N$ builds two lists: ER_i, the list of edges for which u_i is the reporter; and EU_i, the list of edges for which u_i is the updater. At timestep 0, to initialize state, u_i carries out the following steps. If EU_i is not empty, u_i broadcasts its value v_i to its neighbors (specifically for reporters of edges in EU_i). Meanwhile, for each edge $e_{ij} \in \mathsf{ER}_i$, u_i listens for a broadcast message from u_j (the updater of e_{ij}) containing its value v_j; u_i then computes the edge difference $d_{ij} = v_i - v_j$ and records it. Finally, u_i sends a message to the base station containing all its recorded edge differences, together with its own value v_i (if $u_i \in N_m$). At the end of initialization, the base station knows the values of all directly monitored nodes, as well as value differences along all monitored edges.

Continual Phase The continual phase is carried out by all nodes simultaneously at every timestep, without additional intervention by the base station. The frequency of timesteps is a user-controlled parameter. Note the proper choice of this parameter depends on many factors beyond the scope of this paper, such as application-specific measures of information utility, and the amount of time required to finish all communication in each timestep; hence, we will not elaborate on this point further in this paper.

At each timestep, each node u_i obtains its new value v_i^{new} and compares it with the old value v_i^{old} from the previous timestep. If the two differ, u_i broadcasts v_i^{new} to the reporters of edges in EU_i (if $\mathsf{EU}_i \neq \varnothing$), and also sends it to the base station (if u_i is directly monitored). For each edge $e_{ij} \in \mathsf{ER}_i$, u_i updates d_{ij} as follows. If u_i receives a broadcast message from u_j with updated value v_j^{new}, u_i updates d_{ij} to $v_i^{new} - v_j^{new}$; if u_i has not received a broadcast message from u_j by the end of the timestep, u_i infers that v_j remains unchanged, and updates d_{ij} to $d_{ij} - v_i^{old} + v_i^{new}$. At the end of the timestep, for each d_{ij} that has been updated to a different value in this timestep, u_i sends d_{ij} to the base station. As an optimization, all updates from u_i to the root can be grouped into one message. The base station receives update messages and uses them to keep its collection of monitored nodes values and edge differences up-to-date; if a quantity is not updated in a timestep, it is assumed to have remained the same since the previous timestep.

Value Reconstruction The base station can reconstruct the value of any node at any time. Consider node u_i. With a valid CONCH plan, one of the following must be true: (1) u_i is directly monitored; (2) there is a path from some directly monitored node u_{j_0} to u_i, and each edge on the path is monitored. In the first case, the base station should know the value at u_i directly. In the second case, the base station knows the value at u_{j_0} as well as the value differences $d_{j_0 j_1}, d_{j_1 j_2}, \ldots, d_{j_m i}$ along the edges on the path to u_i. Clearly, the value of u_i can be computed by

$$v_{j_0} - \Big(\sum_{0 \leq k < m} d_{j_k j_{k+1}} \Big) - d_{j_m i}.$$

Note that if $d_{j_{k+1}j_k}$ is tracked instead of $d_{j_kj_{k+1}}$, the latter is simply given by $-d_{j_{k+1}j_k}$.

In fact, the base station can reconstruct all values efficiently in one pass using a topological sort of the constraint network starting from the directly monitored node values.

4.2 Cost-Based Construction of CONCH Plan

While any spanning forest can serve as a minimal and valid CONCH plan, it may not necessarily be the best choice in terms of minimizing energy cost. In particular, it is very important to make a clear distinction up front between the CONCH forest (for monitoring) and the routing tree of the sensor network rooted at the base station (for communication). In a routing tree the goal might be for messages to travel from a node to the base station in as few hops as possible. It is quite likely, however, that some edges in the routing tree connect nodes whose values happen to be uncorrelated; such edges would not be good candidates for monitoring. This observation is confirmed by experiments in Section 6. It is also possible for a CONCH forest to be shaped in a way that would be horrible for the purpose of routing, such as the example in Figure 2(b). There is no problem here: CONCH edges are not used for routing; updates are still sent to the base station following shortest paths, or using whatever protocol is supported by the underlying network layer.

In this section, we discuss how to construct a CONCH plan that minimizes the energy cost of monitoring. Intuitively, we want to monitor node values and edge differences that change less, or more precisely, that are cheaper to maintain. We solve the optimization problem in two steps. The first step constructs a min-cost spanning forest, with roots corresponding to monitored nodes and edges corresponding to monitored edges. Here, "cost" is defined as the expected energy cost of monitoring, which is calculated from statistics of past value changes and estimated change reporting costs, including communication reliability. The second steps takes the spanning forest as input, and further decides the roles of reporter and updater for each monitored edge. This time the optimization uses a more detailed communication model, which, for example, accounts for the possibility of combining multiple messages to save per-message overhead. We now discuss these two steps in more detail below.

Step 1: Constructing a Min-Cost CONCH Forest We solve this optimization problem by reformulating it as a problem of finding a min-cost spanning tree (MST) of a graph, where the cost is given by the sum of edge weights in the tree. The resulting MST problem can be solved using standard techniques such as Prim's or Kruskal's algorithms [4].

We start with a graph with all nodes N and communication edges E in the sensor network. We weigh each edge e_{ij} according to the estimated cost of reporting its changes to the base station. This estimate is given by $freq(d_{ij}) \times dist(e_{ij})$, where $freq(d_{ij})$ is the frequency with which d_{ij} changes, and $dist(e_{ij})$ is the number of hops in the shortest path (in a routing tree) from e_{ij} to the base station. We can obtain $dist(e_{ij})$ readily from the network topology; the exact distance is difficult to know outside the routing protocol, and may change over time, but an estimate based on a static topology will likely serve for the purpose of optimization. We defer the discussion of how to obtain $freq(d_{ij})$ to later in this section. It is also possible to integrate reliability into weight calculation, which we do in Section 5.

For each node u_i, we also add an "imaginary" edge connecting it to the base station (just for the purpose of solving this optimization problem), intuitively for capturing the possibility of monitoring u_i directly. This imaginary edge is assigned a weight $freq(v_i) \times dist(u_i)$, where $freq(v_i)$ is the frequency with which v_i changes, and $dist(u_i)$ is the distance from u_i to the base station in hops.

We then find the MST of the constructed graph using standard algorithms such as Kruskal's in time $O(|E| \log |E|)$. To convert the result MST into a CONCH forest, we simply make all nodes incident to the imaginary edges in the MST directly monitored nodes; all non-imaginary edges in the MST become monitored edges.

Step 2: Assigning Updaters and Reporters Intuitively, a good assignment should consider the following optimization opportunities. Consider a monitored edge e_{ij}. First, if v_i changes less frequently than v_j, then it might be better to make u_i the updater, because it requires fewer broadcasts to track the edge difference than the other way around. Second, if u_i is closer to the base station than u_j, then making u_i the reporter might be better because of cheaper reporting. Third, if u_i is incident to more monitored edges than u_j, then making u_i the updater might be better because it can service more edges with just a single broadcast message. Because of the complex interactions among these opportunities and role assignment, it is rather difficult to gauge the quality of heuristic role assignment. Instead, we formulate optimal role assignment as a linear program, and we can guarantee that the assignment will be a factor of 2 within the optimum.

To guide optimization, we again use past observations to approximate future behaviors. In particular, we use a sequence $\mathcal{S}$ of snapshots of the sensor network previously observed at consecutive timesteps, where each snapshot contains all node values in the same timestep. Again, we defer the discussion of how to obtain $\mathcal{S}$ to later in this section. From $\mathcal{S}$ we can easily derive the following information: $change(i,t)$ means that the value at u_i changed in timestep t; $change(i,j,t)$ means that the value difference along e_{ij} changed in timestep t; $freq(u)$ is the total number of times that the value at node u changed in $\mathcal{S}$; $freq(e)$ is the total number of times that the difference along edge e changed in $\mathcal{S}$.

As shorthand notation, let $anc(i,j)$ denote the condition that u_i is a node on the path from u_j to the base station (including u_j itself); let $inc(j,e)$ denote the condition that node u_j is incident to edge e. Given a CONCH forest with monitored nodes N_m and monitored edges E_m, the mixed-integer program assigns the following variables: r_{ei} is set to 1 if u_i is assigned as reporter for edge e, and 0 otherwise; x_{it} is set to 1 if u_i is used for transmitting a message to the base station in timestep t, and 0 otherwise; and y_{it} is set to 1 if u_i must broadcast an update message at timestep t, and 0 otherwise. The program minimizes:

$$(\sigma_s + \sigma_r) \sum_t \sum_i x_{it} + (\delta_s + \delta_r) \sum_{e \in E_m} \sum_{inc(i,e)} dist(i) freq(e) r_{ei} + (\sigma_s + \delta_s) \sum_t \sum_i y_{it} + (\sigma_r + \delta_r) \sum_{e \in E_m} \sum_{inc(i,e)} freq(v_i)(1 - r_{ei})$$

subject to:

$$\forall e \in E_m, u_i \in N, u_j \in N \text{ s.t. } inc(i,e), inc(j,e): \quad r_{ei} + r_{ej} = 1 \tag{1}$$

$$\forall t, u_j \in N_m, u_i \in N \text{ s.t. } change(j,t), anc(i,j): \quad x_{it} \geq 1 \tag{2}$$

$$\forall t, e \in E_m, u_i \in N, u_j \in N \text{ s.t. } change(e,t), inc(j,e), anc(i,j): \quad x_{it} \geq r_{ej} \tag{3}$$

$$\forall t, e \in E_m, u_j \in N : change(j,t), inc(j,e): \quad y_{jt} \geq 1 - r_{ej} \tag{4}$$

Line (1) dictates that each edge has one and only one reporter. Lines (2) and (3) enforce that for each node that is directly monitored or a reporter, all it ancestors (and itself) must communicate in timesteps when some of its monitored quantities change. Line (4) enforces that each node acting as an updater must broadcast in

timesteps when its value changes. The objective function captures parts of the energy cost that are affected by reporter assignment (which do not include the per-byte cost of updating directly monitored values). The idea is to find a reporter assignment that works best for the sample sequence $\mathcal{S}$; if the past is a reasonable predication of the future, we would expect this assignment to work well.

Despite the use of boolean variables, we can solve the above program as a linear one, and simply round fractional solutions to the nearest integer. Because of the linear objective function, the solution is guaranteed to be within a factor 2 of the optimum; in practice, the solution tends to be much better than that.

Discussion Collecting history and statistics for CONCH optimization is straightforward and in a sense comes "for free" because the very goal of the lossless data collection is to record all historical sensor values. Thus, the sample sequence $\mathcal{S}$ used in the second optimization step discussed above can be taken directly from the archived history. Statistics such as *freq*, used also in the first optimization step, can be easily derived from $\mathcal{S}$. If no previous history is available, i.e., CONCH is used on a new deployment, we can simply use any reasonable spanning forest (e.g., the routing tree) as a "tentative" plan. We run this tentative plan for a number of timesteps (e.g., for several days), collect the data, and then use it to find a better CONCH plan. For the purpose of optimization, we do not need the tentative plan to specifically monitor or sample any candidate edge—*any* valid CONCH plan would give us the change characteristics of *all* nodes and *all* edges.

Our description of CONCH assumes central planning at the base station, with periodic re-optimization. We can also make CONCH more dynamic and more responsive to evolving data characteristics, by enabling nodes to locally adjust portions of the plan. For example, an edge reporter can easily swap its role with the updater by keeping a time-weighted measure of how often the updater sends updates, together with a mirrored measure of how often its own value changes. If its own value is changing less frequently than the updater's, then it can initiate a swapping of roles.

As another example, a reporter can locally evaluate how much an edge benefits from spatial suppression. It again maintains statistics, this time on the frequency with which both nodes change values together and the frequency with which at least one node changes. The ratio between the two is a measure of the value of monitoring the edge. If the nodes rarely change together, then the spatial suppression is not beneficial. The reporter may then sever the edge. In that case, both nodes must be directly monitored (since neither knows which is connected to a directly monitored node).

5 Failure-Resilient CONCH

Reliability is a major concern for wireless sensor networks because of high failure rates for both nodes and edges. The main obstacle in CONCH is that it depends heavily on suppression for energy savings. It is difficult to differentiate whether the absence of a message is due to suppression or failure. Because we assume the former, when failure occurs, if a message would otherwise be sent, we will mis-assign a node value or edge difference; constraint chaining may cause such errors to propagate to multiple derived values.

There are two types of failures: permanent node failures, and transient failures causing message loss. To handle permanent node failures, we use the standard heartbeat technique. If a node has suppressed updates and reporting for some number of consecutive timesteps, it sends a heartbeat. If a node fails, neighboring nodes will detect the absence of either standard or heartbeat messages, and report the failure to the base station. In addition, if the edge between a neighbor and the failed node was a monitored edge, the neighbor severs the edge, reports its own value to the base station, and turns itself into a directly monitored node. These local adjustments to the CONCH plan ensure the validity of the plan is maintained (i.e., values at all nodes besides the failed one can still be correctly monitored). The base station has the option of installing a new CONCH plan if the result of local adjustments is inefficient.

In the remainder of this section, we focus on the message loss problem due to transient failures. A communication edge may temporarily fail for a number of reasons, such as when a moving obstacle temporarily blocks the line of sight between two nodes. It does not help to exclude such an edge from a CONCH plan, since we can often expect the edge to be functional again by the time we could make adjustments. Instead, we propose a framework for coping with transient failures, which supports reconstruction of values despite missing updates and removal of data inconsistencies caused by missing updates.

Overview of Strategy CONCH relies on three types of messages for monitoring: (1) those reporting node constraint violations, from a monitored node to the root, (2) those reporting edge constraint violations, from the reporter of a monitored edge to the root, and (3) those updating node values used in maintaining edge constraints, from a monitored edge's updater to its reporter. Transient failures of messages of the third type are relatively cheaper to handle because these messages are single-hop. Here, we use a simple approach that requires an acknowledgment from reporter back to updater; the updater retransmits its message until an acknowledgment is received.

For failures of messages of the first two types, however, this simple approach would be too expensive because these messages may involve many hops. Not only are multi-hop messages and acknowledgments more costly and prone to higher failure rates, the sequential nature of the acknowledgment protocol also implies communication within each timestep can take much longer to finish, which may lead to unacceptably low reporting frequency. Therefore, for failures of messages reporting constraint violations, we propose an alternative strategy that does not require the use of acknowledgments. The remainder of this section discusses this strategy in detail.

Reintroducing Redundancy The key to maintaining CONCH's viability is to build redundancy back into CONCH. In Section 4, we discussed how to build *minimal* CONCH plans to minimize energy consumption. The highly non-redundant nature of such plans, however, makes them extremely susceptible to failures. There are a number of ways to reintroduce redundancy into CONCH plans, and we describe one here.

The idea is to build multiple, different spanning forests over the network. If an edge is used in any of the spanning forests, it is monitored; also, if a node is a root in any of the spanning forests, it is directly monitored. In building these forests, our intuition is to have less reliable edges participate in fewer forests, so the impact of their failures will be minimal. On the other hand, we want more reliable edges to participate in most or all forests, in order to avoid adding more monitored edges where we are confident we can do without. This behavior can be implemented by integrating failure probabilities into edge weights prior to forest construction using MST (cf. Section 4.2). Assume a message for an edge e_{ij} gets lost with probability $fail(e_{ij})$. Suppose $cnt(e_{ij})$ is the number of forests for which e_{ij} has already been chosen. We assign e_{ij} the following weight in constructing a new forest: $freq(d_{ij}) \times dist(e_{ij}) \times (1 + fail(e_{ij}))^{cnt(e_{ij})+1}$. This weight captures two intuitions. First, less reliable edges have higher weights. Second, the weights of less reliable edges rise for each subsequent

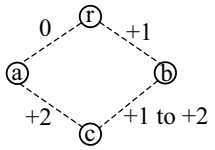

Figure 3: Conflicting solutions for node c.

forest in which they are chosen. In contrast, edges with perfect reliability ($fail(e_{ij}) = 0$) never rise in weight.

This approach assumes independent failure probabilities, and then adds redundancy as needed. Other approaches are possible. For example, with knowledge of joint failure probabilities, we might choose subsets of redundant edges that are unlikely to fail at the same time. If failure is geographically correlated, this approach implies choosing scattered edges. We plan to fully explore construction of non-minimal CONCH plans in future work.

Reconstructing Values Simply incorporating redundancy does not allow correct reconstruction of values in the presence of failures. In fact, while following redundant constraint chains to a particular node, if an error has occurred in one of these chains, we will likely generate conflicting values. Consider the network fragment in Figure 3. It contains one redundant edge, producing two independent paths from a directly monitored node r to another node c in question. The edges are labeled with their last reported differences, one of which changes from $+1$ to $+2$ in the current timestep. In calculating c's value, we derive both $r+2$ and $r+3$. While we know the correct current value of e_{bc}, we have no way of discerning which of the other edges has changed but failed to report and, therefore, no way of choosing the correct solution. It is crucial that we be able to step from simply having redundancy to utilizing it to generate a solution. One option is to generate all possible consistent solutions with associated probabilities. Doing that is difficult, however, because the solution space can be huge.

Instead, we frame the problem as an optimization that finds the assignment of node values with the maximum likelihood, conditioned on all reports received by the root in this timestep, the previous values from the last timestep, as well as any prior knowledge of value change probabilities and message failure rates. To make the optimization problem computationally feasible, however, we have to make some simplifying assumptions. While these assumptions obviously may not hold in practice, they may still lead us to a high-quality (i.e., reasonably likely) assignment of the node values. Specifically, we assume each monitored quantity changes independently with a fixed probability (c_i for node u_i and c_{ij} for edge e_{ij}). We also assume messages reporting constraint violations fail independently with a fixed probability (f_i for u_i reports and f_{ij} for e_{ij} reports).

With these assumptions, we can formulate the optimization problem as a mixed-integer program. Intuitively, the program attempts to set node values as consistently as possible, favoring inconsistencies in constraints that change more and for which reports fail more. In the following, let N_r denote the set of monitored nodes for which the root has received report, and let E_r denote the set of monitored edges for which the root has received report. The program uses the following constants: v_i^{old} denotes the previous (default) value at node u_i; $d_{ij}^{old} = v_i^{old} - v_j^{old}$ denotes the previously known (default) difference along edge e_{ij}; v_i^r denotes the new value for u_i (if received); d_{ij}^r denotes the new edge difference value for e_{ij} (if received). The program sets variables v_i (for each node) and d_{ij} (for each monitored edge), representing its belief of the new values for u_i and e_{ij}, respectively. Boolean variables x_i (for each monitored node without a report) and y_{ij} (for each monitored edge without a report) are set appropriately by the program to indicate whether the respective quantities have changed. The program maximizes:

$$\sum_{u_i \in N_m - N_r} x_i \log \frac{c_i f_i}{1 - c_i} + \sum_{e_{ij} \in E_m - E_r} y_{ij} \log \frac{c_{ij} f_{ij}}{1 - c_{ij}},$$

subject to:

$$\forall u_i \in N_m - N_r : \ (x_i)(\max) \geq |v_i - v_i^{old}| \quad (5)$$

$$\forall e_{ij} \in E_m - E_r : \ (y_{ij})(\max) \geq |d_{ij} - d_{ij}^{old}| \quad (6)$$

$$\forall e_{ij} \in E_m : \ v_i + d_{ij} = v_j \quad (7)$$

$$\forall u_i \in N_r : \ v_i = v_i^r \quad (8)$$

$$\forall e_{ij} \in E_r : \ d_{ij} = d_{ij}^r \quad (9)$$

Lines (5) and (6) (where $\max$ is a large number chosen to be greater than all possible values on the right of the inequalities) appropriately sets the boolean variables. If the calculated node value does not equal its previous value, the corresponding x variable must be set to 1; otherwise, it will be set to 0. The same applies to edges. Line (7) encodes constraint chaining itself; to build a consistent solution, the difference in calculated node values across an edge constraint must be equal to the calculated value assigned to the edge by the program. Lines (8) and (9) state that the calculated value for any node or edge reported in the current timestep must be set to the reported value. Therefore, any solution that violated these would certainly be incorrect. Further, there must exist a feasible assignment with all of these variables fixed. The correctness of reported node values is obvious; the correctness of reported edge values is established by the acknowledgment protocol for updater-reporter communication discussed earlier. This protocol guarantees the reporter sees the correct value at the updater, so it never reports an erroneous edge difference.

The maximization goal, which may appear cryptic at the first glance, effectively maximizes the logarithm of the probability of the solution conditional on the received reports. Given the choice between a reliable, infrequently changing constraint and an unreliable, frequently changing constraint, the program prefers to believe that the latter constraint was violated but the report was lost. Note that independence assumption is needed here to express the goal as a linear objective; otherwise, the objective function will be much more complicated.

The maximum-likelihood approach does have a potential problem. Although it guarantees its solution is the most likely one, the probability that it reflects reality may still be very low because there are so many possibilities. We might have more directed questions, such as what is the probability a particular node has value greater than some amount. The mixed-integer program cannot answer these. We plan to explore alternative approaches that do support these types of questions.

6 Experimental Evaluation

We now evaluate performance of the presented query processing algorithms: temporal suppression, spatial suppression, NEIGHBORHOOD, and CONCH. Spatial is an implementation of [12], with the global ordering concession mentioned in Section 1. For testbed, we use our own simulator of a network of Crossbow MICA2 motes [5], which uses a generic MAC-layer protocol, and models communication as explained in Section 3. The network resides in a rectangular grid. Each grid point represents a square meter and produces some value at each timestep. Nodes are randomly placed on grid points, and takes the values at these grid points as their readings. Node radio range is set at 50 meters. Because we are interested in continuous queries, we ignore start-up costs, which are admittedly

higher for CONCH than simpler query plans, because such costs are amortized over a large number of subsequent timesteps. Hence, in the following, we focus on the average energy (mJ) expended by communication per timestep.

We discretize node values in "tiers." Tier resolution is set to 10 for all experiments, i.e., values in $[0, 10)$ fall into tier 1, values in $[10, 20)$ fall into tier 2, and so on. Node density is set to 1 node per $200m^2$ unless otherwise noted. Each test simulates a different combination of value change and network scenarios. To prime CONCH, we first run each test for a short amount of time with a default spanning forest, and feed the results to CONCH forest construction. We then evaluate performance using the output forest for the remainder of the test.

Density of Sensor Nodes We use a simple density experiment to demonstrate the redundancy problem in NEIGHBORHOOD. In this scenario, all node values are equal throughout, all rising by a single tier every timestep. The number of nodes is fixed, but the area of the network, and therefore density, varies across runs. The results are shown in Figure 4 with a log-scaled y-axis. Most striking in this graph is that NEIGHBORHOOD performs far worse than other algorithms, and its energy consumption increases with density. The reason is that, as density increases, the number of pairs of nodes within communication distance of each other increases. Therefore, average neighborhood size increases, and so does the number of monitored edge constraints. In each timestep, because all nodes change tiers, each node broadcasts its value change, and messages are sent across all edges. While the total number of broadcasts is unaffected by density, the amount of listening that must occur is equal to the number of edges. Listening, therefore, accounts for the increasing energy consumption as density increases. Note since all node values are equal in this experiment, none are ever sent to the root. Otherwise, increasing density might also cause reporting nodes to send increasingly larger reports listing more neighbor values. Due to NEIGHBORHOOD's poor performance, we omit it from future graphs to avoid diluting more interesting results. In contrast, CONCH, which maintains the same number of constraints regardless of density, is unaffected in this experiment. in their reports.

Another interesting feature in the density graph is that spatial suppression performs quite well, and far better than it does in subsequent experiments. Since all nodes have the same tier value, its suppression opportunities are maximized. As we increase density by shrinking area, there are fewer "coverage" nodes, whose reports allows neighboring nodes to suppress, because each coverage node has more neighbors.

Consistently Rising Values This experiment is a simple case where results are predictable. Nodes start with values directly proportional to their distance from the center of the network and all rise at the same rate of one tier every timestep. We vary the size of the range between minimum and maximum node values, and thus the number of tiers; the smaller the range, the fewer the number of tiers, and the more nodes in the same tier. The results are shown in Figure 5.

Temporal suppression performs poorly and is unaffected by the number of tiers. Since nodes always change tiers, all nodes report their values every timestep. Spatial outperforms temporal with only a few tiers. When the number of tiers is low, most nodes suppress themselves because of high likelihood of overhearing a neighbor in the same tier. CONCH outperforms both spatial and temporal. The only energy spent is on updaters broadcasting across their edges to reporters. Since the differences across all monitored edges remain the same from timestep to timestep, the reporters always suppress and no messages are sent to the base station. Finally, we examine NEIGHBORHOOD's performance (not shown in the figure). Like CONCH, it suppresses all reports, but still performs poorly despite this ideal setup. Its excessive edge maintenance consumes as much energy (at all numbers of tiers) as spatial does at 12 tiers.

Varying the number of tiers does affect spatial's performance. Increasing the number of tiers increases the number of unique tier values that any particular node overhears, which in turn decreases the probability that the overheard values average to the node's own tier and thus, the probability of suppression. Since CONCH utilizes spatial suppression, one might expect the number of tiers to impact it as well. Remember, though, that CONCH monitors the differences between nodes. As the number of tiers increases, more neighboring nodes do have differing tier values, but the differences between them remain the same in every timestep, so still no reports are sent.

CONCH completely leverages spatial suppression in this case. All edges have frequencies of change of zero, while nodes change every timestep. Therefore, only edge constraints, and no node constraints, are monitored. In this case, the spanning forest consists of a single spanning tree.

Outliers The next two scenarios examine the impact of outliers. Non-outlier nodes all share the same behavior. In Case 1 non-outliers remain in the same tier over all timesteps. In Case 2 they change tiers every timestep. In both cases, with some probability, a node is chosen to be an outlier, whose value changes drastically and unpredictably each timestep. We vary the probability with which nodes are made outliers across runs. Performance for Case 1 is shown in Figure 6, while Case 2 is shown in Figure 7.

The key difference between the cases is that temporal suppression shifts from performing as well as CONCH in Case 1, where both send more energy as outlier probability increases, to performing much worse in Case 2, where temporal suppression is unaffected by probability. In Case 2, temporal reports every node (whether outlier or not) every timestep, so is not affected by outlier probability. In Case 1, non-outliers never change tiers, so temporal only reports the outliers, exhibiting ideal behavior. CONCH reacts to Case 1 by monitoring the outliers with node constraints, matching the same fundamental behavior of temporal. In Case 2, while temporal suffers, CONCH continues to perform well (although with the added expense of edge updater messages, since nodes change tiers every timestep). CONCH achieves good performance by continuing to leverage spatial suppression by monitoring the edges connecting non-outliers as edge constraints, while monitoring outliers as node constraints.

Wavefront Wavefront simulates waves passing as vertical lines from left to right over the network. The frequency is such that as soon as one wave reaches the right edge of the network, another starts. Wave speed determines how often waves occur. A node's value is tied to the distance covered by the recurring wavefront since it last passed over the node; a node is highest when the wave is over it and lowest just before. We set value range such that there are 4 possible tiers. We vary wave speed across runs. The faster the wave, the more ground it covers between timesteps, so more nodes change tiers each timestep.

Results are shown in Figure 8. As wave speed increases, as expected, temporal consumes more energy. CONCH, on the other hand, continues to consume a low amount of energy; no matter how quickly the wave moves, the number of reported edges is tied to the number of borders, the vertical lines dividing the network into different tiers (4 in this case). Only two types of edges detect changes to their difference calculations, and must report in a particular timestep: those that now cross a border but previously did not,

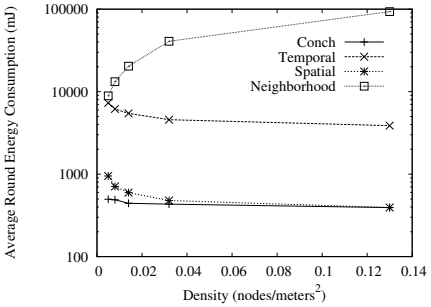

Figure 4: Density of sensor nodes.

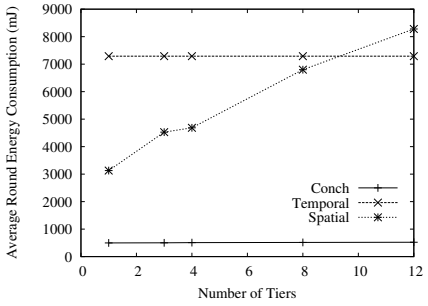

Figure 5: Consistently rising values.

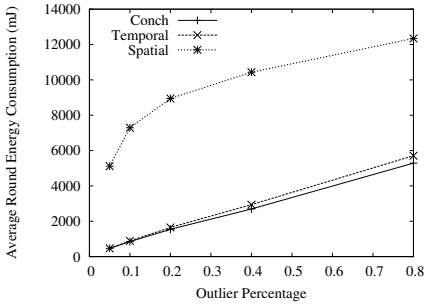

Figure 6: Only outliers change.

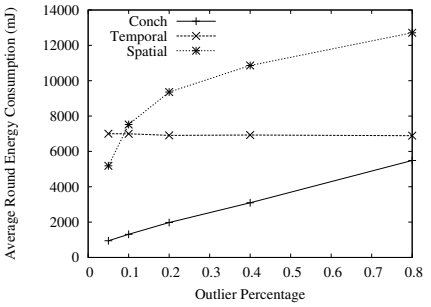

Figure 7: Outliers/non-outliers both change.

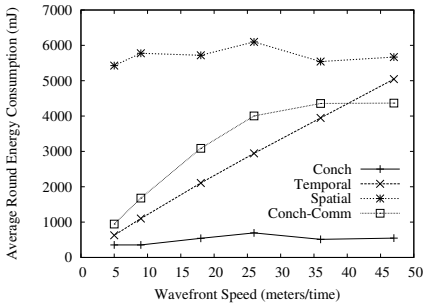

Figure 8: Wavefront.

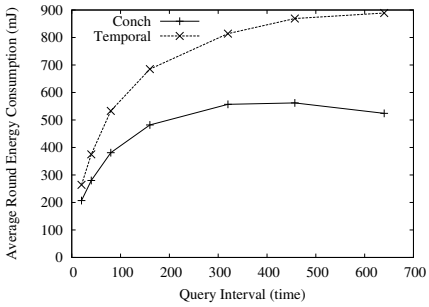

Figure 9: Heat transfer.

and those that now do not cross a border, but previously did. Spatial, because of its lack of memory of prior state, is unaffected by wave speed, and performs worse than the other algorithms simply because too many coverage nodes and nodes along borders must report.

We also include the algorithm CONCH-COMM, which is CONCH but with edge scores based solely on the *dist* (communication distance to base station) function, ignoring *freq*. CONCH greatly outperforms CONCH-COMM, and by an increasing margin as wave speed increases. The key observation is, since the wave is a vertical line, the best edges to monitor are those connecting nodes with small difference in x-coordinates. Monitoring these edges decreases the likelihood the edge intersects a tier border at a given time. If no intersection ever occurs, the edge's nodes always change tiers simultaneously, so the edge constraint is never violated, and no reporting is ever done. The slower the wave moves, the fewer edges chosen by CONCH-COMM are intersected each timestep. As the wave speeds up, the more likely intersections become, up to a plateau. CONCH, on the other hand, expressly seeks out vertical or near-vertical edges that are unlikely to be intersected at any time. This shows the importance of constructing a minimal spanning forest with CONCH scoring, rather than using a naively scored one or simply the routing tree.

The wavefront example's favoritism for vertical CONCH edges provides an opportunity to differentiate the routing tree from the CONCH forest. We depict these for a mini-setup of the wavefront scenario with 20 nodes in Figure 10. The routing tree connects all nodes to the root (in the lower left) in as few hops as possible. There is no preference for edge direction, and we see edges at a variety of angles. The CONCH forest contains mostly vertical or near-vertical edges, with only a few horizontal edges, necessary to connect all nodes.

Heat Transfer This scenario uses a model of heat transfer borrowed from [3]. An event raises the temperature at some grid point. In subsequent timesteps, the heat disperses outward to neighboring points, until the network eventually reaches equilibrium. In each epoch, $T(i,j)$, the temperature at grid point (i,j), is updated using the temperatures from the previous epoch according to the following calculation:

$$T(i,j) \leftarrow T(i,j) + a\Big(T(i+1,j) + T(i-1,j) + T(i,j+1) + T(i,j-1) - 4T(i,j)\Big).$$

Here, $a \leq 0.25$ is a dispersion factor. We experiment with heat transfer by varying the length of a timestep (time between queries) as measured by the number of epochs. The more time lapses, the more heat transfers without detection, and the more change occurs from the previous sensor reading.

We find spatial suppression, in addition to NEIGHBORHOOD, is dramatically more expensive than temporal and CONCH. To avoid diluting the most interesting result, we plot only these last two in Figure 9. As expected, both algorithms spend more energy the more time lapses between queries. When the change between consecutive timesteps is small, they perform similarly. With longer time between queries, changes accumulate more. The cost of temporal increases at a much higher rate because more nodes report, but CONCH increases slowly. As in the wavefront scenario, CONCH prefers to monitor edges whose nodes are affected by heat transfer in the same way and at the same time. In this scenario, it favors edges between nodes that are equidistant from the heat source. Both algorithms' slopes of increasing consumption begin to level out when query interval is very long, because the number of nodes exhibiting change each query update increases slowly by this point.

Experiments Involving Failures

We next evaluate the approach to handling failures presented in Section 5 with a 100-node network. Our goal is to test the viability of the approach for recovering a reasonably accurate solution in the presence of failures. To that end, we build a simple CONCH tree containing just a single tree root, and incorporate redundancy by

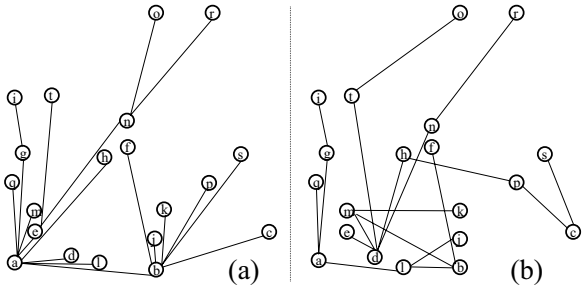

Figure 10: Routing vs. CONCH trees.

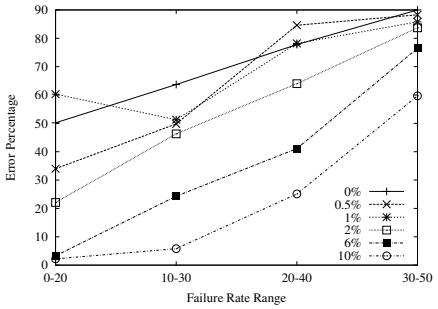

Figure 11: Random walk with failures.

adding additional edges to monitor at random. The number of additional edges is chosen as a percentage of the number of possible remaining edges that can be monitored. Therefore, 0% leaves the CONCH tree as is, while 100% turns the CONCH tree into NEIGHBORHOOD, with all possible edges monitored. As we will see from experiment results, the effectiveness of low-percentage redundancy points to the overkill in NEIGHBORHOOD's redundancy. We generate failures as follows. Given a failure rate range, say 30%–50%, we pick a rate for each monitored quantity at random from this range, and subject reporting messages for this quantity to this failure rate.

Random Walk The first reliability experiment is a random walk scenario with density set to 1 node per $600m^2$. All grid points begin with equal values. At each timestep, the new value at a grid point is drawn from a normal distribution centered at the previous value at this grid point. There is no spatial correlation among values in this scenario, as in wavefront or heat transfer, and so CONCH does not provide great benefit. We mainly use this scenario to produce a test where many reports are needed to produce the correct solution, rather than just few ones. If only one report is needed in a round, for example, we either get 0% or 100% accuracy for the round, depending on if that report is successfully transmitted. With random walk, we see a smooth interplay between failure rates, redundancy, and solution error. We graph failure rate ranges versus average error (percentage of node values that are not correctly assigned by our approach per timestep), and plot results for a series of CONCH plans, each built with differing amounts of redundancy. We run the random walks for 10 timesteps, starting after an initial timestep when all constraints try to report. Results are shown in Figure 11, and depict two major trends: First, the higher the failure rate, the higher the error; second, the higher the redundancy rate, the lower the error.

Heat Transfer We next return to the heat transfer scenario to see the impact of redundancy under conditions favorable to CONCH. Density is set higher at 1 node per $56m^2$. In the base case of neither failure nor redundancy, few reports are sent to derive all node values. We expect, then, to be "lucky" some timesteps and receive all of those reports, and "unlucky" other timesteps, where even one failed report results in many miscalculated values. To investigate this phenomenon, we make two plots. Figure 12 shows performance of different redundancy levels for increasing failure rates. Figure 13 shows the cumulative distribution of error rate in each timestep (query rounds) during a run, for a number of redundancy levels. That is, for a particular error percentage e on the x-axis, the y-coordinate of a point is the percentage of query rounds with error rate no more than e. Failure rate for Figure 13 is fixed in the range 30%–50%. The cumulative distribution shows that the lucky/unlucky phenomenon occurs at low redundancy levels, with some rounds at low and some at high error levels. This phenomenon is pronounced at low redundancy levels by constraints that change infrequently, which our approach tends to trust, but may have failed at some point. In subsequent rounds, until the constraint changes again, the reported value continues to be incorrect, with few chances to counter it. The phenomenon diminishes at higher redundancy levels, where error rate is low for all rounds.

Finally, we examine the cost of redundancy. Additional redundancy means more constraints potentially report. In contrast, failure means less reports reach the root. Using the heat transfer scenario, Figure 14 plots average number of constraint-update reports received at the root versus average error rate. The varying number of reports is the result of running CONCH at different redundancy levels, labeled on the plot. Failure rate is fixed at 20%–40%. As expected, as redundancy level increases, the number of reports increases, and error decreases. Temporal suppression appears as a point. For temporal, we assume that nodes fail to report with 30% probability. In that case, given a 100-node network, 70 nodes report and average error is 30%. CONCH is immediately more accurate than temporal with far fewer reports. The 0%-failure (and 0%-redundancy) case is drawn as a vertical line, which serves as a base case for comparison. At 0% redundancy CONCH sends fewer reports than the base case but, of course, with a sacrifice in accuracy. As the number of reports meets and exceeds the baseline, accuracy improves. The number of reports necessary to push error near 0% is quite a bit higher than the base case, demonstrating redundancy is not close to free. This finding serves as motivation to do a thorough investigation of what to monitor redundantly in order to push error suitably low, while at the same time minimizing the overhead.

In the scenarios we have tested, failure rates are assigned in a fairly tight range. In practice, we might expect reports for some constraints to fail with high likelihood (due to location of the node, for example) and others to be very reliable. If so, it should be easier for our integer-program approach to decide which constraints to dismiss when resolving conflicts. More thorough evaluation is needed to verify this conjecture.

In summary, our experiments involving failures show that we can effectively leverage the extra information provided by redundancy to set node values with some accuracy. Nevertheless, we pay a price for redundancy by monitoring more constraints. Further, as failure rate increases, our ability to recover values accurately decreases. Therefore, redundancy is a viable technique for coping with failures, but more work is needed in this area.

7 Conclusion and Future Work

We have investigated using temporal and spatial suppression for the problem of continuously collecting all readings from a sensor

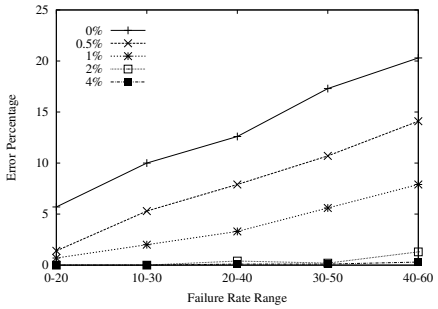

Figure 12: Heat transfer with failures.

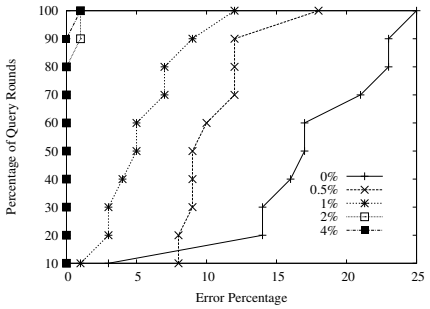

Figure 13: Error CDF; heat transfer.

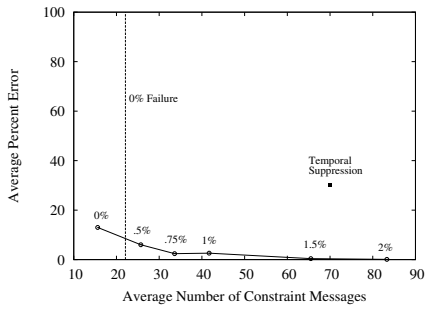

Figure 14: Redundancy cost; heat transfer.

network. Our suppression techniques can benefit many monitoring scenarios, where change is slow or predictable, and data is spatially correlated. We develop an effective monitoring policy combining both suppression types. We present NEIGHBORHOOD as a first attempt at using this policy, but show its redundancy hampers its usefulness. Progressing from there, minimal CONCH uses a spanning forest that monitors the minimum number of constraints possible, solving the redundancy problem. We present techniques for building a spanning forest with minimal monitoring costs. Then, to cope with failures, we add redundancy back into CONCH, and develop a method for correctly interpreting results affected by failure, but bolstered by redundancy. Our experimental results show CONCH indeed performs as well as or outperforms other algorithms in most cases, degrading gracefully to temporal suppression in specific cases where edge monitoring has no advantage. The results also show that it is feasible to use redundancy to compensate for failure and recover node values with reasonable accuracy.

In the future we plan on advancing CONCH in two directions. The first is to maintain constraints of increasing complexity, involving more larger spatial models that move beyond pair-wise monitoring. The second is to further improve our failure-handling strategy, by developing better techniques for selectively introducing redundancy to maximize its benefit, and for reasoning with data containing uncertainty that arises from failures.

8 References

[1] K. Chintalapudi and R. Govindan. Localized edge detection in sensor fields. In *Proc. of the 2003 IEEE Sensor Network Protocols and Applications*, May 2003.

[2] D. Chu, A. Deshpande, J. Hellerstein, and W. Hong. Approximate data collection in sensor networks using probabilistic models. In *Proc. of the 2006 Intl. Conf. on Data Engineering*, Apr. 2006.

[3] Chuck Conner. Modeling Heat Transfer in Parallel. `http://www.cas.usf.edu/~cconnor/parallel/2dheat/2dheat.html`.

[4] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*, chapter 23. McGraw-Hill/MIT Press, 2001.

[5] Crossbow Inc. *MPR-Mote Processor Radio Board User's Manual*.

[6] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Compressing historical information in sensor networks. In *Proc. of the 2004 ACM SIGMOD Intl. Conf. on Management of Data*, June 2004.

[7] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *ACM/IEEE Trans. on Networking*, 11(1):2–16, 2002.

[8] A. Jain, E. Chang, and Y. Wang. Adaptive stream resource management using kalman filters. In *Proc. of the 2004 ACM SIGMOD Intl. Conf. on Management of Data*, June 2004.

[9] Y. Kotidis. Snapshot queries: Towards data-centric sensor networks. In *Proc. of the 2005 Intl. Conf. on Data Engineering*, Apr. 2005.

[10] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *Proc. of the 2002 USENIX Symp. on Operating Systems Design and Implementation*, Dec. 2002.

[11] A. Manjhi, S. Nath, and P. Gibbons. Tributaries and deltas: Efficient and robust aggregation in sensor network streams. In *Proc. of the 2005 ACM SIGMOD Intl. Conf. on Management of Data*, June 2005.

[12] X. Meng, L. Li, T. Nandagopal, and S. Lu. Event contour: An efficient and robust mechanism for tasks in sensor networks. Technical report, UCLA, 2004.

[13] S. Pattem, B. Krishnamachari, and R. Govindan. The impact of spatial correlation on routing with compression in wireless sensor networks. In *Proc. of the 2004 Intl. Conf. on Information Processing in Sensor Networks*, Apr. 2004.

[14] D. Petrovic, R. Shah, K. Ramchandran, and J. Rabaey. Data funneling: Routing with aggregation and compression for wireless sensor networks. In *Proc. of the 2003 IEEE Sensor Network Protocols and Applications*, May 2003.

[15] G. Pottie and W. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[16] M. Sharaf, J. Beaver, A. Labrinidis, and P. Chryanthis. Tina: A scheme for temporal coherency-aware in-network aggregation. In *Proc. of the 2003 ACM Workshop on Data Engineering for Wireless and Mobile Access*, Sept. 2003.

[17] I. Solis and K. Obraczka. Efficient continuous mapping in sensor networks using isolines. In *Proc. of the 2005 Mobiquitous*, July 2005.

[18] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proc. of the 2003 ACM Conf. on Embedded Networked Sensor Systems*, Nov. 2003.

[19] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *ACM SIGMOD Record*, 31(3), 2002.

[20] Z. Zhou, S. Das, and H. Gupta. Connected k-coverage problem in sensor networks. In *Proc. of the 2004 IEEE Intl. Conf. on Computer Communications and Networks*, Oct. 2004.