Failure-Aware Cascaded Suppression in Wireless Sensor Networks

Yi Zhang, Kristian Lum, and Jun Yang

Abstract—Wireless sensor networks are widely used to continuously collect data from the environment. Because of energy constraints on battery-powered nodes, it is critical to minimize communication. *Suppression* has been proposed as a way to reduce communication by using predictive models to *suppress* reporting of predictable data. However, in the presence of communication failures, missing data is difficult to interpret because it could have been either suppressed or lost in transmission. There is no existing solution for handling failures for general, spatiotemporal suppression that uses *cascading*. While cascading further reduces communication, it makes failure handling difficult, because nodes can act on incomplete or incorrect information and in turn affect other nodes. We propose a cascaded suppression framework that exploits both temporal and spatial data correlation to reduce communication, and applies coding theory and Bayesian inference to recover missing data resulted from suppression and communication failures. Experiment results show that cascaded suppression significantly reduces communication cost and improves missing data recovery compared to existing approaches.

Index Terms—Spatiotemporal suppression, wireless sensor networks, coding theory

1 Introduction

Wireless sensor networks have been widely used to monitor and study the environment [1], [2]. In many applications, sensor nodes sample at a given frequency and report all collected data to a base station for further analysis (without strict real-time requirements). A major constraint in designing many such networks is energy. Sensor nodes, often running on batteries, are expected to operate for a long time in order to minimize the labor cost of maintenance and intrusiveness to the environment. Conventional wisdom is that wireless communication tends to dominate energy consumption on sensor nodes, so reducing communication is key to designing energy-efficient sensor applications.

Reducing Communication Various techniques have been proposed to reduce communication in sensor networks. One of the proposed approaches is *suppression* [3], [4], which is based on the idea that most sensor readings are "predictable" and therefore need not be reported. In suppression, both the sender and the receiver agree on a predictive model for the data being monitored. The sender can *suppress* the transmission of a reading if and only if its value can be predicted by the model within a prescribed error bound. Barring the possibility of transmission failure, the receiver would know that the value of a missing report lies within the error bound around the model-predicted value. A simple example is the *value-based temporal suppression* [4], where the model always predicts the current value to be the same as the last transmitted value. With this scheme, a reading is reported only if it differs from the last transmitted value by more than some threshold ϵ. Clearly, the amount of communication saved depends on how well the model predicts data. Fortunately, many measurements of the environment (e.g., temperature, humidity) do change in predictable ways, making suppression effective in reducing communication.

For greater reduction in communication, we need to exploit data correlation more aggressively than temporal suppression. Most environmental variables exhibit correlation not only in time but also in space. For instance, two nearby nodes often have identical or similar temperature readings. Without knowing the perfect correlation model, *spatiotemporal suppression* is a realistic way of exploiting such correlation and further reducing communication. For example, in *Ken* [3], suppression based on *disjoint-cliques models* works by collecting multiple, distributed sensor readings at a *clique root*, which then uses a spatiotemporal model on these readings to decide which ones (if any) to report to the base station.

Cascaded Suppression We consider a natural technique called *cascading* to extend suppression. The idea is that a node uses suppression in reporting its readings to another node, which then uses suppression again in further reporting this reading together with others to a third node, etc. The following example illustrates a *two-tier cascaded suppression* scheme, or *CS2* for short.

Example 1. *We group nodes into clusters according to spatial vicinity, as illustrated in Figure 1. Each cluster has an elected* head; *others are called* child *nodes. Child nodes in a cluster report to their head, which then exploits the spatiotemporal correlation among the received readings to suppress reporting*

- Y. Zhang and J. Yang are with the Department of Computer Science, Duke University, Durham, NC 27708. Email: {yizhang, junyang}@cs.duke.edu
- K. Lum is with the Institute of Mathematics at UFRJ, Brazil. Email: kristian@dme.ufrj.br

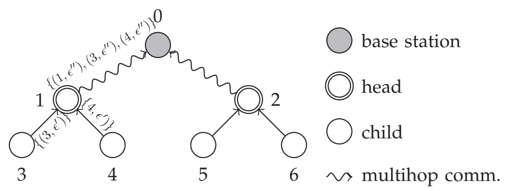


Fig. 1. Two-tier cascaded suppression.

to the base station.[1] *For example, for the cluster headed by node 1, both node 1 and the base station maintain a model for the readings of nodes 1, 3, and 4. If the current values for the cluster can be predicted to within some error bound ($\pm\epsilon''$), the head suppresses transmission; otherwise, it selects a subset of values to transmit so that the others can be predicted conditioned on the transmitted.*

Within each cluster, between every child and the head, temporal suppression is applied. A child does not have to always transmit; instead, they can agree on a temporal model for the child's reading, and the child can suppress a report to the head if its value can be predicted to within $\pm\epsilon'$. Assuming reliable communication, the base station can bound all sensor readings to within $\pm\epsilon$, where $\epsilon = \epsilon' + \epsilon''$. Now, we have a two-tier cascaded suppression scheme.

Previously proposed suppression schemes can be seen as special cases of cascaded suppression. For example, value-based temporal suppression is obtained by letting each node form its own cluster. With the disjoint-cliques model of Ken [3], cluster members report to their heads in every timestep; therefore, this scheme can be seen as a special case of CS2 with $\epsilon' = 0$ and $\epsilon'' = \epsilon$. The generality of cascaded suppression, as we shall see in Section 8, can lead to greater communication savings.

Coping with Failures So far, cascaded suppression seems like a simple generalization of suppression. Once we consider the reality of transient message failures, however, things become more challenging. Failures introduce ambiguity into suppression. If a receiver receives nothing, it cannot tell whether the missing data was due to suppression or failure. Depending on which case is true, we have completely opposite interpretation of the missing data. If a suppression occurred, the actual value should be well predicted by the model; if it was a failure, the actual value would be outside the error bound of the prediction. Hence, without addressing failures, suppression will have little practical use.

How can we deal with failures in suppression? A common practice is *Automatic Repeat reQuest* (*ARQ*), where receivers reply to successfully received messages with ACKs, and senders retransmit up to a number of times or until they receive ACKs. However, ACKs and retransmissions consume extra energy. Also, they still cannot resolve ambiguity—a missing message might still be caused by the failure of multiple retransmissions.

1. A head need not be within direct communication distance from the base station; messages between them can go through multiple nodes. Figure 1 is a conceptual view of the suppression hierarchy, not the network topology.

TABLE 1
A possible outcome of cascaded suppression with transmission failures. Wrong values/bounds are boxed.

Time	Node 3's reading	$3 \rightarrow 1$	Node 1's prediction	$1 \rightarrow 0$	Bound at base
1	$x^{(1)}$	$\surd$	$x^{(1)}$	$\surd$	$x^{(1)} \pm \epsilon'$
2	$x^{(2)}$	$\perp$	$x^{(1)}$	$\perp$	$x^{(1)} \pm (\epsilon' + \epsilon'')$
3	$x^{(3)}$	$\times$	$\boxed{x^{(1)}}$	$\perp$	$\boxed{x^{(1)} \pm (\epsilon' + \epsilon'')}$
4	$x^{(4)}$	$\perp$	$\boxed{x^{(1)}}$	$\perp$	$\boxed{x^{(1)} \pm (\epsilon' + \epsilon'')}$
5	$x^{(5)}$	$\surd$	$x^{(5)}$	$\surd$	$x^{(5)} \pm \epsilon'$

BaySail [4] proposes an alternative to ARQ based on the following observation. Knowing just the fact that a transmission has been attempted (even without knowing its actual content) is sometimes enough for the purpose of recovering data. To this end, BaySail lets the sender in a temporal suppression scheme inject redundancy into its messages in the form of the timestamps of its last few transmissions. Using this information, the receiver (base station) can retroactively reconstruct the history of attempted transmissions from the sender, and use this history to help recover missing data.

Failure handling for cascaded suppression, however, is fundamentally more challenging. Note that the receiver reconstructs the correct transmission history only retroactively. This issue does not pose a problem for BaySail, where suppression is not cascaded. With cascading, unfortunately, the receiver of a reading x may act as sender in another suppression setup, and it may act (i.e., decide what to report) based on its current guess of x, which may turn out later to be wrong. This intricate chain effect is illustrated below.

Example 2. *Consider again Example 1. Table 1 gives an example of how node 3's reading are reported. We denote a successful transmission by $\surd$, a failure by $\times$, and a suppression by $\perp$.*

At time 3, node 3 realizes its current value $x^{(3)}$ differs from its last transmitted value $x^{(1)}$ by more than ϵ', so it transmits $x^{(3)}$ to node 1, but the message is lost. At this point, node 1 is unaware of the attempted transmission. Thinking that the missing value as suppression, node 1 now has a wrong guess of node 3's reading.

In timesteps 3 and 4, node 1 acts on this wrong guess, incorrectly concluding that it need not report readings of either node 3 or 4 to the base station. At this point, the bounds derived by the base station on node 3's reading may be wrong.

Finally, at time 5, node 1 receives a message from node 3 and discovers the previous failure at time 3 (recall that node 3 "piggybacks" on each message the timestamps of its last few transmissions). Node 1 now knows that it had wrong values of node 3 in timesteps 3 and 4, but it has already made suppression decisions based these wrong values. Should it now somehow reconcile with the base station to undo these mistakes? If so, how? And what if some messages for reconciliation fail as well?

Contributions Designing a failure-aware cascaded suppression framework is challenging in many ways. How do we systematically cope with cases where a single failure has a rippling effect on other readings and nodes? What information is necessary for interpreting missing data resulted from failures and cascaded suppression? How should that information be communicated efficiently? What suppression models can we use to capture spatiotemporal correlations, without making failure handling and data analysis intractable?

In this paper, we provide a holistic solution to these questions. We propose a general framework for cascaded suppression that is both communication-efficient and failure-resilient. Inspired by prior studies on spatiotemporal suppression and on the Bayesian approach to handling failures, our work not only goes beyond them in terms of their respective strengths, but also provides a solution that unifies them for the first time. We believe our work is an important step toward making suppression a practical paradigm for data collection in wireless sensor networks.

More specifically, we show that cascaded suppression beats previously proposed suppression techniques (namely, disjoint-cliques of Ken [3] and value-based temporal suppression in BaySail [4]) in terms of flexibility of control and the ability to exploit spatiotemporal correlations in sensor data to reduce communication.

Another compelling advantage of our solution is the principled approach towards failure handling. The only existing solution with this feature works only for simple temporal suppression [4], and it deals with none of the intricacies that arise with cascading. We show how to resolve the problem of nodes acting on inaccurate information—not by scrambling for corrective actions, but by carefully logging and forwarding information that will allow the base station to reconstruct history and interpret data later.

To communicate such information efficiently, we further apply the idea of *convolutional coding* [5] from coding theory. Using a novel decoding technique tailored to our setting, our coding-based redundancy scheme holds a clear advantage over BaySail's method.

Finally, we evaluate our solution in terms of the energy cost and the quality of information recovered on both synthetic and real-world datasets. By quantifying the energy cost of computation and comparing it with that of communication—an aspect often overlooked by previous work on suppression—we provide additional insight on the choices of suppression algorithms and cluster sizes. Compared with previous work, our solution offers better tradeoffs between cost and quality of data collection.

2 Related Work

Many approaches have been proposed to reduce communication in sensor networks. *Tiny AGgregation* (*TAG*) [6] utilizes in-network processing to aggregate data as it travels toward the base station. However, TAG is not suited for raw data collection. *BBQ* [7] proposes probabilistic model-driven data acquisition. Queries about sensor data are answered by consulting a correlation-aware statistical model. If the model cannot provide results with enough confidence, the base station acquires readings from a subset of nodes in order to reach the desired confidence level. However, the model must be trustworthy; otherwise, an answer could be wrong and there is no way of knowing it.

The idea of model-based suppression has been applied in *Ken* [3], which uses dynamic, spatiotemporal probabilistic models. Our solution differs from Ken in three significant ways. First, as discussed in Section 1, our cascaded suppression is more general and creates more opportunities for reducing communication. Second, Ken recovers data in the form of deterministic bounds, whereas we combine information obtained from the suppression scheme with statistical models to recover data in the form of posterior distributions. Third, Ken does not directly address the issue of transient communication failures. The Markovian nature of their models only guarantees that, once a new value for a reading arrives at the base station, the models are synchronized with respect to this reading. However, nothing can be said about the data during the time when the base station receives nothing. This issue weakens the data quality guarantee offered by Ken.

While one can add temporal suppression between individual cluster members and cluster heads in Ken, doing so without properly handling failures will further weaken Ken's data quality guarantee—now that cluster heads can have incorrect values for cluster members, values received by the base station may be incorrect! This issue highlights the need for coping with failures.

PAQ [8] is another example of utilizing multivariate time-series models to reduce communication. Each node builds an autoregressive model for predicting local readings. When needed, nodes can send outlier readings or updates to model parameters. Like Ken, PAQ does not cascade models or handle transient communication failures; a lost update can mislead the base station.

Compression is an alternative approach to saving communication. From a theoretical perspective, Slepian and Wolf have shown that multiple sources can be coded with rate less than or equal to their joint entropy, even without communication among them [9]. To achieve this bound, *distributed compression* (*source coding*) [10], [11] has been proposed. However, there is a strong assumption: the joint distribution quantifying the correlations of the sources must be known. In many wireless sensor networks, this assumption does not hold; learning correlation structures is often one of the purposes of deploying a sensor network.

As shown in [12], with packet loss, compression algorithms such as LZW, LZ77, and adaptive Huffman Coding perform even worse than not using compression at all. This problem is caused by the dependency on previously decompressed data for decompressing subsequent data. The authors propose a fault-tolerant algorithm, which tries to keep the dictionaries used by senders and receivers synchronized. However, it cannot interpret lost packets or recover missing information.

BaySail [4] is a framework for resolving uncertainty caused by failures in suppression. It relies on Bayesian inference to reconstruct missing data using the knowledge of the suppression schemes and the redundancy injected by senders. On a high level, our approach towards failure handling is similar. The key difference is that BaySail considers only temporal suppression between a node and the base station. We tackle the problem for cascaded spatiotemporal suppression, which is more general and efficient than temporal suppression. Also, as motivated in Section 1, failure handling for cascaded suppression is fundamentally more challenging—more nodes are involved (because of spatial aspect of suppression), and some nodes may act on inaccurate information and in turn affect others (because of cascading).

The idea of injecting redundancy has been combined with explicit retransmission requests in [13]. Bloom filters are used to encode past transmission timestamps. However, Bloom filters introduce false positives; a suppression can be misidentified as a failure. To reduce false positives, Bloom filters need more bits. The authors suggest 10 bits per transmission timestamp, which may not be better than sending the timestamps themselves. Also, it is unclear how to interpret data if the retransmission requests/replies are themselves lost. Finally, like BaySail, this approach does not address the complexity of failure handling for cascaded suppression.

3 Problem and Framework Overview

Consider a wireless sensor network of N nodes (numbered with $0, \ldots, N-1$) with sensors attached. For simplicity of exposition, assume a single sensor per node; it is easy to extend our results to cases where a node has multiple sensors. We assume each node takes readings at predetermined timesteps. The reading taken by node k at timestep t is denoted by $x_k^{(t)}$. The network is responsible for reporting each $x_k^{(t)}$ to a base station node (designated 0), such that the base station, barring transmission failures, knows $x_k^{(t)}$ to within $\pm\epsilon$ when timestep t ends.

The wireless network is multi-hop and unreliable; a message can travel through multiple nodes to reach its destination and can fail in the process. Without additional mechanisms such as ACK, we do not assume that the destination node knows about the failed attempt, or that the source node knows its message has failed. Due to unreliable communication, it is inevitable that the base station may not know every sensor reading to within $\pm\epsilon$.

Typically, the goal of environmental sensing is twofold. Suppose that the sequence of sensor readings $\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(t)}$ is governed by a model $M(\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(t)}; \mathbf{\Theta})$, where $\mathbf{\Theta}$ denotes the model parameters (unknown or with some prior). We want to 1) recover any missing readings as accurately as possible, and 2) learn about $\mathbf{\Theta}$ from the information received by the base station.

Our goal is to devise a method for reporting data to the base station, so that we can support the two tasks above while minimizing communication, in the presence of possible transmission failures.

We present a framework for failure-aware cascaded suppression, which enables an end-to-end solution to the problem above. Our framework comprises four components. 1) A cascaded *suppression scheme* is used to minimize communication cost. 2) To cope with transient message failures in such a cascaded setting, a *redundancy scheme* is used to resolve ambiguities caused by failures. 3) *Constraint derivation* techniques convert basic facts obtained from the two schemes into hard constraints relating missing and received data, which are then fed as input to 4) a *Bayesian inference* algorithm to produce estimates of missing data and model parameters.

Previous work on BaySail [4] and Ken [3] both fit in our framework and can be considered as special cases of our method. BaySail tackles failures but only handles temporal suppression; Ken performs spatiotemporal suppression but does not cope with failures. Our results go beyond the mere combination of prior arts, however. As discussed in Sections 1, failure handling for cascaded suppression is fundamentally more difficult than that for temporal suppression, because of the possibility of nodes acting on incorrect information. Moreover, cascaded suppression is more general than the disjoint-cliques model of Ken, and performs better in our experiments in Section 8. Finally, we also show how to apply forward error correction in this novel setting.

The next four sections discuss the four components of our framework in turn. To make the discussion more concrete, we present a specific instantiation, *CS2* (*2-tier Cascaded Suppression*, illustrated in Figure 1), as a running example.

4 Suppression Scheme

On a high level, a suppression scheme is a communication-efficient way of delivering sensor readings to the base station, such that the base station, assuming no communication failures, knows every reading to within $\pm\epsilon$ of its true value. We model a *cascaded suppression scheme* as an edge-labeled directed graph over the N nodes in the network. Each edge $i \to j$ in this graph, called a *suppression edge*, is labeled with a set of sensor-bound pairs $K^{ij} \subseteq \{0, \ldots, N-1\} \times \mathbb{R}^+$. This suppression edge represents a reporting contract from node i to node j. Specifically, for each pair $(k, \epsilon_k) \in K^{ij}$, i is responsible for reporting to j such that at each time t, the value of the reading of sensor k known at i, denoted by $x_k^{(t,i)}$, and the value known at j, denoted by $x_k^{(t,j)}$, differ by no more than ϵ_k (assuming no transmission failures): $|x_k^{(t,i)} - x_k^{(t,j)}| \leq \epsilon_k$. For example, we have labeled some suppression edges in Figure 1. A *valid* suppression scheme has the property that for each sensor k, all edges whose labels contain k form a single, acyclic path from node k to the base station;

furthermore, all ϵ_k's in the labels on this path sum up to no more than ϵ. Intuitively, nodes on this path relay the reading of sensor k to the base station via a chain of suppression-based reporting contracts.

We first describe generally how the sender and the receiver fulfill the suppression contract, and then present a specific solution used by CS2. We begin by introducing some notation and terminology. For each suppression edge $(i \to j; K^{ij})$:

- $\mathbf{x}^{(t,i)}, \mathbf{x}^{(t,j)}$ denote the *values known at sender i and j at time t*; they are vectors of size $|K^{ij}|$, with one component per sensor.
- $\mathbf{z}^{(t,i)}$ denotes the *transmission bit vector at time t*, which represents the subset of $\mathbf{x}^{(t,i)}$ transmitted from i to j at time t. The bit $z_k^{(t,i)}$ corresponding to sensor k is 1 if $x_k^{(t,i)}$ is transmitted, or 0 otherwise. The transmitted subset of values is then $\mathbf{x}^{(t,i)}[\mathbf{z}^{(t,i)}]$.
- $\mathbf{s}^{(t,i)}$ denotes the *suppression state*, a subset of all values ever transmitted by i to j, i.e., $\mathbf{s}^{(t,i)} \subseteq \bigcup_{t' \le t} \mathbf{x}^{(t',i)}[\mathbf{z}^{(t',i)}]$. The choice of this subset depends on the actual suppression scheme. For each value we also track the node ID and the time. $\mathbf{s}^{(t,i)}$ is replicated at both i and j.
- U^{ij} denotes the *suppression state update function*, which updates the suppression state as $\mathbf{s}^{(t,i)} \leftarrow U^{ij}(\mathbf{s}^{(t-1,i)}, \mathbf{x}^{(t,i)}[\mathbf{z}^{(t,i)}])$. U^{ij} is replicated at both i and j.
- F^{ij} denotes the *prediction function*, which predicts $\mathbf{x}^{(t,i)}$ using the previous suppression state and the current transmitted subset: $F^{ij}(\mathbf{s}^{(t-1,i)}, \mathbf{x}^{(t,i)}[\mathbf{z}^{(t,i)}])$. F^{ij} is replicated at both i and j.
- $\boldsymbol{\epsilon}^{ij}$ denotes the *vector of bounds* associated with this suppression edge, where each component ϵ_k^{ij} denotes the bound associated with sensor k; i.e., $(k, \epsilon_k^{ij}) \in K^{ij}$.

To simply notation, we shall omit node identity superscripts (i and j) when there is no ambiguity.

We use $\preceq$ to denote the following relationship between two vectors $\mathbf{a}$ and $\mathbf{b}$: $\mathbf{a} \preceq \mathbf{b}$ iff $\forall i\, (|a_i| \le b_i)$. The invariant maintained by the suppression edge $i \to j$ is thus $\mathbf{x}^{(t,i)} - \mathbf{x}^{(t,j)} \preceq \boldsymbol{\epsilon}$.

To establish a suppression edge, sender i and receiver j need to agree on U and F; i initializes the suppression state by sending all readings for K to j in the first timestep. In each subsequent timestep t, sender i performs the following steps:

1) *Determine the transmission bit vector* $\mathbf{z}^{(t)}$, such that $F(\mathbf{s}^{(t-1)}, \mathbf{x}^{(t,i)}[\mathbf{z}^{(t)}]) - \mathbf{x}^{(t,i)} \preceq \boldsymbol{\epsilon}$. We prefer transmitting as few values as possible. In particular, we set $\mathbf{z}^{(t)} = \mathbf{0}$ and transmit nothing if $F(\mathbf{s}^{(t-1)}, \emptyset) - \mathbf{x}^{(t,i)} \preceq \boldsymbol{\epsilon}$, i.e., all new values can be predicted to within $\pm\boldsymbol{\epsilon}$ without transmitting anything. To the other extreme (the worst case), we may need to set $\mathbf{z}^{(t)} = \mathbf{1}$ to transmit all readings.
2) *Update the suppression state*: $\mathbf{s}^{(t)} \leftarrow U(\mathbf{s}^{(t-1)}, \mathbf{x}^{(t,i)}[\mathbf{z}^{(t)}])$.

Receiver j performs the following steps:

1) *Make a prediction*: If j receives $\mathbf{x}^{(t,i)}[\mathbf{z}^{(t)}]$ from i, set $\mathbf{x}^{(t,j)} \leftarrow F(\mathbf{s}^{(t-1)}, \mathbf{x}^{(t,i)}[\mathbf{z}^{(t)}])$; otherwise, $\mathbf{x}^{(t,j)} \leftarrow F(\mathbf{s}^{(t-1)}, \emptyset)$.
2) *Update the suppression state*: If j receives $\mathbf{x}^{(t,i)}[\mathbf{z}^{(t)}]$, set $\mathbf{s}^{(t)} \leftarrow U(\mathbf{s}^{(t-1)}, \mathbf{x}^{(t,i)}[\mathbf{z}^{(t)}])$; otherwise, $\mathbf{s}^{(t)} \leftarrow U(\mathbf{s}^{(t-1)}, \emptyset)$.

It is easy to verify the above protocol maintains the invariant $\mathbf{x}^{(t,i)} - \mathbf{x}^{(t,j)} \preceq \boldsymbol{\epsilon}$ in the absence of failures.

The prediction function F need not be perfect, because the sender can always transmit readings (in the worse case, all of them) to make the prediction bounded. Hence, the correctness of suppression does not depend on the accuracy of F. Nevertheless, better F reduces communication.

At the level of an individual suppression edge, the above scheme operates similarly as other spatiotemporal suppression techniques such as Ken [3]. The difference, which creates potential for higher efficiency, comes from chaining suppression edges together into a cascaded suppression scheme. Here, the receiver j of a suppression edge may in turn serve as a sender for another suppression edge; hence, $\mathbf{x}^{(t,j)}$, j's bounded approximation of $\mathbf{x}^{(t,i)}$, may be used to drive further reporting from j.

4.1 Suppression Scheme in CS2

We consider how to implement the general cascaded suppression scheme for CS2, where nodes are grouped into clusters. CS2 employs two types of suppression edges: those from a child in a cluster to the cluster head, and those from a cluster head to the base station. The first type uses simple value-based temporal suppression, and the second type is based on multivariate statistical models. Suppose the base station wants to know every $x_k^{(t)}$ within $\pm\epsilon$. To this end, we label all child-to-head suppression edges with bound ϵ', and all head-to-base suppression edges with bound ϵ'', such that $\epsilon = \epsilon' + \epsilon''$.

We now discuss the two types of edges in more detail. For child-to-head suppression edges, it is straightforward to cast value-based temporal suppression in terms of the general description of a suppression edge introduced earlier in this section. Therefore, we focus on head-to-base suppression edges below.

Consider a suppression edge $(h \to b; K)$ from a cluster head h to the base station b, where K includes a (k, ϵ'') for each sensor k in the cluster headed by h. The idea is to exploit the spatiotemporal correlations among sensor readings within a cluster in order to reduce h's communication to b. To this end, we capture these correlations with a first-order vector autoregressive model (VAR(1)), and use it to predict current readings from previously transmitted ones. Specifically, the model $\pi(\mathbf{X}^{(t)}, \mathbf{X}^{(t-1)}; \mathbf{A}_\pi, \mathbf{c}_\pi, \boldsymbol{\Sigma}_\pi)$ is given by:

$$\mathbf{X}^{(t)} = \mathbf{A}_\pi \mathbf{X}^{(t-1)} + \mathbf{c}_\pi + \boldsymbol{\varepsilon}^{(t)}, \qquad (1)$$

where $\mathbf{X}^{(t)}$ denotes a vector of random variables representing the readings of sensors in K known to h at time t (i.e., $\{x_k^{(t,h)} \mid (k, \cdot) \in K\}$ can be viewed as an observation of $\mathbf{X}^{(t)}$), $\mathbf{A}_\pi$ is a $|K| \times |K|$ matrix, $\mathbf{c}_\pi$ is constant vector of size $|K|$, and $\boldsymbol{\varepsilon}^{(t)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_\pi)$ is a random noise vector following a multivariate normal distribution (MVN) with zero mean and covariance matrix $\boldsymbol{\Sigma}_\pi$. Spatial correlation is introduced by $\boldsymbol{\varepsilon}^{(t)}$ and is further propagated via $\mathbf{A}_\pi$, because a single component of $\mathbf{X}^{(t)}$ depends on all components from the previous time. $\mathbf{A}_\pi$, $\mathbf{c}_\pi$, and $\boldsymbol{\Sigma}_\pi$ can be obtained from data collected from pilot runs and periodically updated.

We let the suppression state contain the last transmitted values for each node in the cluster. With the VAR(1) model in (1), the prediction function computes the expectation for the untransmitted components of $\mathbf{X}^{(t)}$ conditioned on their last transmitted values ($\mathbf{s}^{(t-1)}$) and the currently transmitted components. Formally,

$$F(\mathbf{s}^{(t-1)}, \mathbf{x}^{(t,h)}[\mathbf{z}^{(t)}]) = E(\mathbf{X}^{(t)} \mid \mathbf{s}^{(t-1)}, \mathbf{x}^{(t,h)}[\mathbf{z}^{(t)}]).$$

For an example of how suppression state is maintained and the prediction function F is computed in CS2, consider a cluster with four sensors numbered 1 to 4. Figure 2 shows an example of how the suppression state evolves. At time t (Figure 2a), the previous suppression state $\mathbf{s}^{(t-1)}$ consists of the last transmitted values $x_1^{(t-2,h)}$, $x_2^{(t-1,h)}$, $x_3^{(t-1,h)}$, and $x_4^{(t-3,h)}$. Then, at time $t+1$ (Figure 2b), after h transmits $x_2^{(t,h)}$, the suppression state $\mathbf{s}^{(t)}$ becomes $x_1^{(t-2,h)}$, $x_2^{(t,h)}$, $x_3^{(t-1,h)}$, and $x_4^{(t-3,h)}$. At time t, h computes F as

$$E\left(X_1^{(t)}, X_2^{(t)}, X_3^{(t)}, X_4^{(t)} \mid x_1^{(t-2,h)}, x_2^{(t-1,h)}, x_3^{(t-1,h)}, x_4^{(t-3,h)}\right)$$

and compares them with the actual values $x_1^{(t,h)}, x_2^{(t,h)}, x_3^{(t,h)}, x_4^{(t,h)}$. Suppose the predictions are not accurate enough, and h finds that

$$E\left(X_1^{(t)}, X_3^{(t)}, X_4^{(t)} \mid x_1^{(t-2,h)}, x_2^{(t,h)}, x_3^{(t-1,h)}, x_4^{(t-3,h)}\right)$$

are now within bounds of the actual values $x_1^{(t,h)}$, $x_3^{(t,h)}$, and $x_4^{(t,h)}$. Hence, h transmits only $x_2^{(t,h)}$.

To compute the prediction function F, it suffices to maintain the joint distribution $p(\mathbf{X}^{(t)}, \mathbf{s}^{(t-1)})$ at each timestep t, by using the following recursion

$$p(\mathbf{X}^{(t)}, \mathbf{s}^{(t-1)}) = \int_{\mathbf{X}^{(t-1)}} p(\mathbf{X}^{(t-1)}, \mathbf{s}^{(t-1)}) p(\mathbf{X}^{(t)} | \mathbf{X}^{(t-1)}). \tag{2}$$

Note $p(\mathbf{X}^{(t-1)}, \mathbf{s}^{(t-1)})$ can be obtained by a marginalization of $p(\mathbf{X}^{(t-1)}, \mathbf{s}^{(t-2)})$, because $\mathbf{s}^{(t-1)}$ is always a subset of $\mathbf{X}^{(t-1)} \cup \mathbf{s}^{(t-2)}$. $p(\mathbf{X}^{(t)} | \mathbf{X}^{(t-1)})$ is available from (1).

Because of the VAR(1) model we choose, all distributions involved above are MVN. Thus all operations reduce to standard matrix calculations that can run on sensor nodes.

Selecting Subset to Transmit One remaining issue is how a cluster head selects a subset from the $|K|$ current readings to transmit if transmitting nothing will make prediction go out of bounds. Selecting the "best" subset is NP-hard, so we consider greedy algorithms that can be implemented easily on sensor nodes. We present two such algorithms, both of which greedily expand the chosen subset one reading at a time until the prediction is bounded. For pseudocode, see Algorithm 1.

- The **quadratic** algorithm evaluates F O($|K|^2$) times. In each iteration it chooses the remaining sensor reading whose addition to the subset would result in the least overall prediction error.
- The **linear** algorithm evaluates F O($|K|$) times. It first sorts all sensor readings in decreasing order of errors in their predictions (assuming nothing is transmitted). Then, it adds the readings one at the time in this order.

Algorithm 1: Greedy algorithms for subset selection.

```
Input: current readings $\mathbf{x}$, suppression state $\mathbf{s}$, error bounds $\boldsymbol{\epsilon}$
Output: $\mathbf{z}$, bitmap representing the subset to be transmitted

Option 1: quadratic algorithm
$\mathbf{z} \leftarrow \mathbf{0}$;
while true do
    if $F(\mathbf{s}, \mathbf{x}[\mathbf{z}]) - \mathbf{x} \preceq \boldsymbol{\epsilon}$ then
        return $\mathbf{z}$;
    $e_{\min} \leftarrow \infty$;
    foreach $k$ where $z_k = 0$ do
        $\mathbf{z}' \leftarrow \mathbf{z}$; $z'_k \leftarrow 1$;
        $\boldsymbol{\epsilon}' \leftarrow F(\mathbf{s}, \mathbf{x}[\mathbf{z}']) - \mathbf{x}$;
        if $e_{\min} > \|\boldsymbol{\epsilon}'\|$ then
            $e_{\min} \leftarrow \|\boldsymbol{\epsilon}'\|$;
            $k_{\min} \leftarrow k$;
    $z_{k_{\min}} \leftarrow 1$;

Option 2: linear algorithm
$\mathbf{z} \leftarrow \mathbf{0}$;
sort components in $\|F(\mathbf{s}, \emptyset) - \mathbf{x}\|$
in descending order;
$\mathbf{p} \leftarrow$ sorted component indices;
foreach $k$ in $\mathbf{p}$ do
    $z_k \leftarrow 1$;
    $\boldsymbol{\epsilon}' \leftarrow F(\mathbf{s}, \mathbf{x}[\mathbf{z}]) - \mathbf{x}$;
    if $\boldsymbol{\epsilon}' \preceq \boldsymbol{\epsilon}$ then return $\mathbf{z}$;
```

Computation Cost Conventional wisdom is that computation costs on sensor nodes are dwarfed by communication costs. Much of the previous work on suppression provides no quantified comparison: PAQ [8] has only coarse estimation of the complexity of updating individual nodes' suppression models, while Ken [3] completely ignores the computation cost of spatial suppression. To better understand the computation cost of suppression, we take a closer look at CS2. Computation costs of child-to-head suppression edges are negligible because value-based temporal suppression can be implemented by a handful of simple instructions. The base station is not resource constrained so computation there can also be ignored. Most of the computational burden is placed on cluster heads in the form of matrix operations. For a cluster of m nodes, at each timestep, it takes $\Theta(m^2(d+m))$ basic operations (e.g., scalar multiplication or addition) to update the suppression model according to (2), where d is the dimension of the model at the previous timestep. The initial prediction, $F(\mathbf{s}^{(t-1)}, \emptyset)$, requires $\Theta(m^3)$ operations. To select a subset to transmit, the quadratic algorithm needs $\Theta(km^4)$ operations, where k is the number of readings selected. The linear

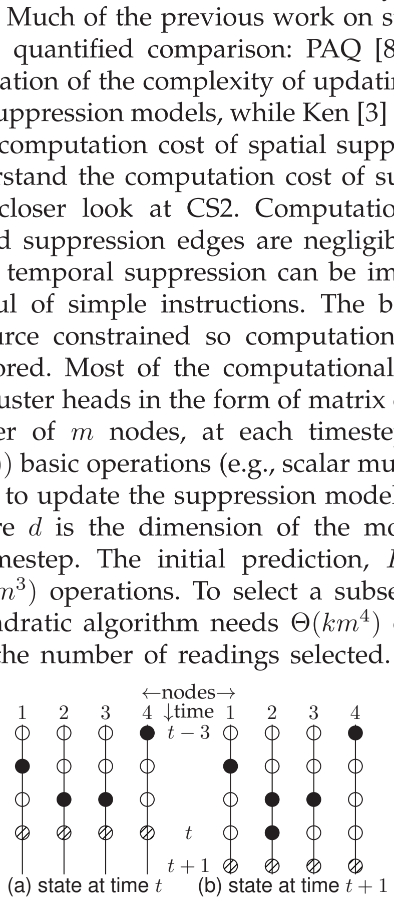


Fig. 2. Suppression state (shared by a cluster head and the base station) as time progresses. Shaded circles: current readings. Filled circles: previously transmitted values.

algorithm needs $\Theta(km^3)$ operations. With large clusters, computation can translate into significant fractions of the total energy cost. Therefore, computation costs must be carefully weighed against communication savings when designing a suppression scheme. Section 8 presents some results of this comparison and discusses its implication in choosing cluster sizes and subset selection algorithms.

Coping with Topology Changes We distinguish two types of topology changes—in the underlying network topology and in the suppression topology. Details about coping with these changes can be found in the online supplemental materials accompanying this paper.

5 Redundancy Scheme

As discussed in Section 1, transient message failure spells trouble for suppression schemes. Ignoring the possibility of failures in interpreting data leads to wrong answers, because lost messages cause the sender and the receiver's suppression states to diverge, leading to meaningless predictions. Acknowledging the possibility of failures, on the other hand, makes it impossible to obtain any bounds whenever the receiver receives no message. In this section, we discuss how senders can piggyback additional information (redundancy) on their messages, such that receivers can construct relevant transmission history.

A straightforward way of coping with failures is *ARQ* (*Automatic Repeat reQuest*), where the receiver explicitly acknowledges a message with an ACK, and the sender retransmits up to some number of times or until it receives an ACK. This solution is inadequate in our setting for two reasons. First, ACKs and retransmissions consume more energy. The problem is aggravated by the high message failure rates in wireless sensor networks (which increase the number of retransmissions) and possibly long distances spanned by suppression edges (e.g., those between cluster heads and the base station, which increase the unit cost of retransmissions and ACKs). Second, although ARQ increases the probability of a message reaching its destination, there is no guarantee that it will. Therefore, we are still unable to infer any bounds on missing readings, thereby losing a key advantage of suppression. In wireless sensor networks, failures in consecutive retransmissions may be correlated, so it is impossible to tell how many retransmissions are required such that we can safely ignore the possibility of failed messages. As we will see in the experiments of Section 8, even with strong assumptions (independent failures) and careful optimization (broadcast of ACKs within clusters), ARQ is not competitive with the techniques we are about to propose.

Towards a Better Solution For simple temporal suppression between a node and the base station, BaySail [4] proposes that the sender attaches to every outgoing message the timestamps of its last r transmissions. Upon successfully receiving a message from this sender, the base station can (retroactively) construct the sender's transmission history, as illustrated by the following example.

Example 3. *Suppose that the base station heard from a node at time 0 and time 3. Before hearing from this node again, the base station has no way of knowing whether the node has transmitted or suppressed for any timestep after 3. Suppose it is now at time 12 and the base station receives another message from the node, together with two transmission timestamps 6 and 8 (in this case the redundancy level $r = 2$). With this information, the base station can reconstruct the transmission history as follows, where $\surd$, $\times$, and $\perp$ indicate successful transmission, failed transmission, and suppression, respectively. Note, however, that the activities at timesteps 4 and 5 remain unknown (indicated by ?).*

$$\begin{array}{ccccccccccccc}
t=0 & & & 3 & & & 6 & & 8 & & & & 12 \\
\hline
\multicolumn{1}{|c|}{\surd} & \multicolumn{1}{c|}{\perp} & \multicolumn{1}{c|}{\perp} & \multicolumn{1}{c|}{\surd} & \multicolumn{1}{c|}{?} & \multicolumn{1}{c|}{?} & \multicolumn{1}{c|}{\times} & \multicolumn{1}{c|}{\perp} & \multicolumn{1}{c|}{\times} & \multicolumn{1}{c|}{\perp} & \multicolumn{1}{c|}{\perp} & \multicolumn{1}{c|}{\perp} & \multicolumn{1}{c|}{\surd} \\
\hline
\end{array}$$

It has been demonstrated in [4] that, for the purpose of statistically reconstructing missing data and learning process parameters from received data, it suffices to know the transmission history; insisting that every non-suppressed reading be received by the base station is often an overkill. The intuition is that the transmission history allows us to establish constraints among missing and received data, which improve the effectiveness of statistical inference. In the example above, the base station can infer, for example, that all values between timesteps 8 and 12 (non-inclusive) are within $\pm\epsilon$ of the value at time 8, but the value at time 12 is not.

The general ideas of adding redundancy to help reconstruct transmission history and converting this history into constraints for inference are still applicable. However, the multivariate and cascaded nature of our suppression scheme poses new challenges.

First, for spatiotemporal suppression involving a vector of readings, simply sending the timestamps of a past transmissions is not enough. Recall that a prediction function takes as input the subset of readings transmitted. Without knowing the membership of this subset, we cannot write a constraint that relates the prediction result to the variables used in the prediction.

Second, because of cascaded suppression, the base station needs to establish a chain of constraints relating the value of a reading known at the base station to the actual value at its source. Each constraint stems from a suppression edge. Thus, the base station needs to reconstruct transmission history not only for nodes that directly report to it, but also nodes that are further upstream. However, the redundancy injected by nodes that directly report to it does not carry any information about upstream nodes' transmissions.

Finally, as illustrated by Example 2, a node may retroactively discover that it has made incorrect assumption about an upstream node. However, at this point, it has already acted on this assumption when deciding whether and what to report to its downstream node.

To address these issues, we first give a solution that extends the timestamp repetition scheme of BaySail.

Then, we discuss how to adapt forward error correction to improve the solution.

5.1 Baseline Solution

The solution we propose is based on *transmission history maps*, which generalize the transmission history illustrated in Example 3. The transmission history map of a given suppression edge tracks, for each timestep, whether there has been a successful transmission, failed transmission, or suppression, or it is unknown from the receiver's perspective. Also, for each (known) attempted transmission (successful or not) at time t, this map keeps track of the transmission bit vector $\mathbf{z}^{(t)}$ (defined in Section 4).

The goal of the redundancy scheme is to enable the base station to reconstruct the transmission history map for every suppression edge. To this end, we operate as follows for every suppression edge $i \rightarrow j$. There are three types of redundancy reporting.

Type-S(ender) To every transmission from i to j, i attaches the last r_S transmission timestamps, together with the transmission bit vector at each of these timesteps. The *type-S redundancy level*, r_S, is a user-specified parameter. This redundancy allows j to build the transmission history map for $i \rightarrow j$.

Type-R(eceiver) If j is not the base station, then j is responsible for keeping the base station updated with the transmission history map for $i \rightarrow j$.[2] There are many ways to accomplish this goal with interesting tradeoffs. We describe one technique below, which we have implemented for the experiments in Section 8.

A transmission history map can be partitioned into *segments*. Each segment begins with a known attempted transmission, and ends right before the next such transmission. It is not difficult to see that the map entries between them must all be either suppression or unknown. A segment g can be represented by a tuple $\langle t_{\text{begin}}, t_{\text{end}}, \mathbf{z}, b_{\surd|\times}, b_{\perp|?}\rangle$, where $[t_{\text{begin}}, t_{\text{end}})$ is the time interval associated with g, $\mathbf{z}$ is the transmission bit map at time t_{begin}, $b_{\surd|\times}$ is a bit indicating whether the transmission at t_{begin} was successful, and $b_{\perp|?}$ is a bit indicating whether entries in $(t_{\text{begin}}, t_{\text{end}})$ are suppression or unknown. Whenever j sends out a message in the direction of the base station, j attaches to that message the last r_R segments of the transmission history map. The *type-R redundancy level*, r_R, is a user-specified parameter. Also, whenever j discovers a new transmission failure from i (using type-S redundancy), j will send type-R redundancy, without piggybacking on another message if necessary.

If j is the target of multiple suppression edges, j is responsible for maintaining one map for each such edge and reporting suppression segments for all such maps.

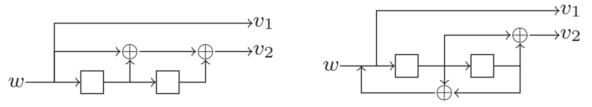
Type-F(orwarder) Finally, if a node n receives a message with type-R redundancy for suppression edge $i \rightarrow j$ and n is not the base station, n is responsible for forwarding this information to the base station. An alternative to simple forwarding is for n to use this information to incrementally reconstruct the transmission history map for $i \rightarrow j$; then, a method similar to type-R redundancy keeps the base station updated with this map.

5.2 Forward Error Correction

The baseline solution above essentially repeats the same data in different messages, where parameters such as r_S control the number of repetitions. Simple repetition is not the smartest way to get recoverability, however. We turn to coding theory for a better solution. In particular, we apply a type of forward error correction called *convolutional coding* [5]. Our new application setting requires retooling of the standard convolutional coding techniques; we discuss the differences at the end of this subsection.

To simplify presentation, consider the problem of sending a sequence of messages of L bits each from one node to another. We assume the receiver either receives each message correctly without any bit error or eventually finds out that it was lost (or corrupted beyond repair). In practice, we can ensure this assumption holds by including checksums and sequence numbers in messages. After first describing our solution to this problem, we will show how to apply it to the more specific scenario of encoding redundancies for cascaded suppression.

Example 4. *Consider a rate $1/2$ convolutional encoder in Figure 3(a). It has two memory registers ($\square$), and two adders ($\oplus$) that perform bit-wise modulo-2 additions. Inputs, outputs, and memory registers all have L bits each. When a new input w enters the encoder, the adders generate two outputs v_1 and v_2 (hence the $1/2$ rate), and then the input is shifted into the memory registers, as directed by the arrows. The content in the rightmost register is evicted after each shift. Figure 3(b) is a different rate $1/2$ encoder with a feedback structure.*



(a) without feedback (b) with feedback

Fig. 3. Two convolutional encoders of rate $1/2$.

To transmit a message m, the sender feeds it to a rate $1/r_C$ convolutional encoder ($r_C = 2$ in the above example), and then transmits the r_C outputs as one message

2. One might wonder why j cannot simply notify the base station of just the failures, i.e., the timestamps of failed transmissions from i and the transmission bit vectors at those timestamps. Unfortunately this information is insufficient. Consider again Example 3, but assume that instead of the base station, it is some other node j that is deciphering the transmission history from i and trying to keep the base station up to date with this history. At time 12, j detects the failures at 6 and 8. If j tells the base station only about these failures, the base station would not know the history during $[0, 3]$. Therefore, for this interval, the base station cannot bound the difference between the true values of the readings at i and the values believed in by j (which were used in its reporting to the base station). In other words, even though j itself knows that this difference is bounded, it has to somehow inform the base station of this fact.

instead of m. Upon receiving the coded message, the receiver uses a decoder to obtain the original input to the encoder. An ancillary data structure the decoder uses is the *trellis diagram*. The *state* of an encoder at a particular time is just the concatenation of its memory register contents. The trellis diagram for an encoder consists of a grid of nodes, arranged in 2^n rows and infinite number of columns, where n is the number of bits in an encoder state. Columns from left to right correspond to time, which advances by 1 on each new input. All nodes in row i represent the i-th possible state of the encoder. An edge with label in/out from a node x in column t to another node y in column $(t+1)$ means the following: if the state of the encoder at time t is as specified by node x, and the input to the encoder is in, then the encoder outputs out and the state transitions to y. For example, Figure 4 shows the trellis diagram for the encoder in Figure 3. Inputs and registers are assumed to be 1 bit each in this example. Only some edge labels are shown. Dashed edges are those labeled with input 0, while solid edges are those labeled with input 1.

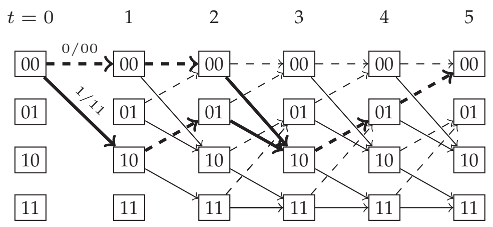


Fig. 4. Trellis diagram for the encoder in Figure 3.

With the trellis diagram, the decoder performs decoding by finding, roughly, the path through the trellis diagram that matches the received sequence of coded messages. More precisely, however, we are looking not for a path per se, but instead the correct input at each timestep. This subtle difference is illustrated by an example. Suppose the encoder in Figure 3 is employed by a sender. The encoder's memory registers are initialized to 00 at time 0. At the receiver's side, suppose the decoder sees a coded sequence ?? ?? ?? 01 01, where ?? denotes a lost message. There are two paths matching this output sequence, marked bold in Figure 4. The top path takes input sequence 0 0 1 0 0 and produces output sequence 00 00 11 01 01, while the bottom path takes 1 0 1 0 0 and produces 11 01 10 01 01. Although the decoder is unable to determine which is the "true" path given only the received sequence, it is able to determine all the input messages except the first one, simply by picking common parts of the two possible input sequences. The actual algorithm can be found in the online supplemental materials accompanying this paper.

Discussion Why does convolutional coding beat repeating messages? Intuitively, a rate $1/r_C$ convolutional encoder can capture in a coded message of size r_C information derived from more than r_C input messages; In contrast, message repetition includes only r_C input messages in the same amount of space. For example, recall the earlier problem of decoding ?? ?? ?? 01 01. Receiving only 2 out of 5 coded messages, we were able to recover the last 4 of the 5 input messages. However, using message repetition, where a message of the same size (2 bits) includes the last input in addition to the current one, receiving the last 2 of the 5 such messages will only allow us to recover the last 3 input messages.

It is worth noting that our application of convolutional coding departs from its standard usage (e.g., Viterbi [14] and Fano [15] decoding). Standard usage typically assumes a channel model and performs probabilistic decoding. Unfortunately, transient message failures in sensor networks remain poorly understood and difficult to model because of the complex interplay of hardware and software issues and (often unpredictable) environmental factors. Therefore, we have taken a different and more conservative approach, making no assumption about the channel model and recovering only inputs that we are absolutely certain of. The result is robust and particularly suitable for the suppression scenario. A more detailed discussion can be found in the online supplemental materials accompanying this paper.

Improving the Baseline Solution There is flexibility in applying our coding-based technique to improve the baseline redundancy scheme in Section 5.1. We can use different types of convolutional coders (see Section 8 for more examples). We can apply coding to entire messages, or just to selected types of information. As an example of the latter, we can change the implementation of type-S redundancy as follows. Instead of including the timestamps and transmission bit vectors of the last r_R transmissions in the current message, we can first feed the very last transmission timestamp and bit vector as input to a rate $1/r_C$ encoder, and then attach the output to the current message. The size of the resulting message remains the same if $r_R = r_C$. Section 8 explores this and other ways of improving the baseline scheme.

5.3 Redundancy Scheme in CS2

Redundancy scheme in CS2 operates exactly as the baseline scheme in Section 5.1, and can be readily improved by the above coding-based technique. The simpler structure of CS2 implies two simplifications, however. First, assuming single sensor per node, for a child-to-leader suppression edge, we do not need to include any transmission bit vector, because it has only one bit that will be always be 1 for every transmission. Second, with two tiers only, there is no type-F redundancy. A child is responsible only for type-S redundancy. A cluster head is responsible for 1) type-S redundancy for its suppression edge to the base station, and 2) type-R redundancy for suppression edges from the children in its cluster.

6 Constraint Derivation

The redundancy scheme in Section 5 allows the base station to construct the transmission history map for every suppression edge. Because of failures, there is

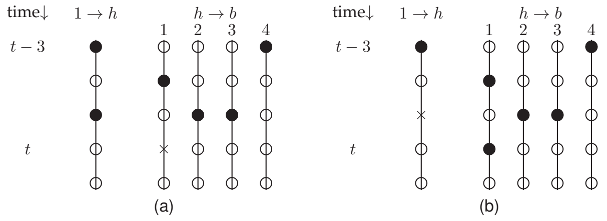


Fig. 5. An example of transmission history in CS2.

no guarantee that the base station will obtain complete maps, but with appropriate amount of redundancy, the hope is that these maps will be mostly complete. Also, note that failures can only affect the completeness of the maps, not the correctness of them.

The transmission history maps, in turn, enable the base station to reconstruct the sequences of actions taken at each node and the rationale behind them. Using the knowledge of the suppression scheme, the base station can derive hard constraints among values of readings known at different nodes and across time. These constraints can then be incorporated in inference. We illustrate how to derive constraints from transmission history maps with a concrete example in the context of CS2.

6.1 Constraint Derivation in CS2

Consider Figure 5(a), which illustrates the transmissions from node 1 to its cluster head h, and from h to the base station b over time. Filled circles denote successfully received values. Suppose all suppression edges are labeled with bound $\epsilon/2$. Let us walk through an exercise of deriving constraints on $x_1^{(t,1)}$, the reading of node 1 at time t. As shown in the figure, b notices that h received a message from node 1 at time $t-1$, followed by suppressions at t and $t+1$. Hence, we can relate the values known at h to the true values at node 1:

$$\left|x_1^{(t,1)} - x_1^{(t-1,1)}\right| \le \frac{\epsilon}{2}, \quad \left|x_1^{(t+1,1)} - x_1^{(t-1,1)}\right| \le \frac{\epsilon}{2}, \tag{3a}$$

$$x_1^{(t,h)} = x_1^{(t+1,h)} = x_1^{(t-1,1)}. \tag{3b}$$

Also, b discovers (by checking redundancy in a later message) a lost message from h at time t, which should have contained the value $x_1^{(t,h)}$ (marked by $\times$ in the figure). Then, at time $t+1$, h suppressed transmission. Therefore, $|x_1^{(t+1,h)} - E| \le \epsilon/2$, where E, the predicted value, is a conditional expectation that turns out to be a linear combination of $x_1^{(t,h)}$, $x_2^{(t-1,h)}$, $x_3^{(t-1,h)}$, and $x_4^{(t-3,h)}$. The latter three were successfully transmitted to the base station, while the lost $x_1^{(t,h)}$ remains a mystery. However, if we combine this inequality with (3), we get linear inequalities involving only unknowns $x_1^{(t-1,1)}$, $x_1^{(t,1)}$ and $x_1^{(t+1,1)}$, the actual readings we care about.

Recall from Section 1 the possibility of nodes acting on incorrect information in cascaded suppression. We now illustrate how we cope with this issue. Consider again the previous example, but suppose instead that the transmission from node 1 to head h at time $t-1$ failed, as shown in Figure 5(b). We need to derive constraints differently. When h transmitted $x_1^{(t,h)}$ at time t, it had not yet discovered the failure of the message from node 1 at $t-1$ (which can be discovered only after a successful transmission from node 1 later). Thus, h's view of x_1 became incorrect starting from $t-1$; therefore, $x_1^{(t,h)}$ and $x_1^{(t+1,h)}$ do not satisfy (3b). Unfortunately, unaware of this discrepancy, h made a decision about what to transmit to b at t based on incorrect information. Although we could let h take corrective actions as soon as it discovers an earlier mistake, doing so incurs additional communication but cannot recuperate the cost of the earlier mistake that has already been paid. Moreover, what if communication also fails during these corrective actions? Instead, we take a simpler approach that avoids expensive (and unreliable) runtime corrections. The key is that *we derive constraints based on the knowledge a node had at the time when it made its decision*. Continuing with the example, b can deduce that $x_1^{(t,h)}$ and $x_1^{(t+1,h)}$ are actually based on node 1's last successful transmission before time $t-1$, which is at time $t-3$. Therefore, we have $x_1^{(t,h)} = x_1^{(t+1,h)} = x_1^{(t-3,1)}$ instead of (3b). Critical to this approach is that the transmission history maps allow us to "repeat history" at the receivers.

Constraints can be similarly obtained for other scenarios. Note that for some cases the obtained constraints are not linear. For example, the fact that the failed message at time t contains $x_1^{(t,h)}$ means if $x_1^{(t,h)}$ is not transmitted, its prediction would have been out of the prescribed bound. Thus a constraint of the form $|\cdot| > \cdot$ is obtained, which is not linear, but an OR of two linear constraints. We currently do not make use of these non-linear constraints due to inference difficulties (although other techniques like the *direction bits* [4] can be used to linearize these constraints at the cost of more communication).

Picking a suppression scheme based on VAR(1) allows us to avoid complex constraints and stay mostly linear; this choice greatly simplifies the next step—inference.

7 Bayesian Inference

Suppose that a model $M(\mathbf{X}^{(t)}, \mathbf{\Theta})$ describes the process governing the sensor readings. Recall from Section 3 our ultimate goals are to 1) recover missing data as accurately as possible, and 2) learn about the model parameters $\mathbf{\Theta}$. Besides sensor readings received by the base station, several other sources of information should be incorporated into inference: 1) redundancy received by the base station, 2) knowledge of the suppression scheme, and 3) any prior knowledge about the model parameters. Section 6 has discussed how to convert (1) and (2) into constraints on missing and received data. With these constraints and any prior knowledge on $\mathbf{\Theta}$, we compute the *posterior* distribution of the missing data and the model parameters via a Bayesian approach.

The posterior may be complex, making analytical integration difficult. Thus, we take a sampling approach. We draw samples from the joint distribution; samples from the marginal of a particular parameter or missing data are obtained by ignoring other components in the joint samples. In contrast to point estimates, sampling allows assessing the uncertainty in the estimates.

Below is a concrete example of how we use Gibbs sampling [16] to recover missing data and learn parameters of a VAR(1) model in the context of CS2.

7.1 Bayesian Inference in CS2

Suppose we wish to model collected data using the following VAR(1) with an exponential covariance structure:

$$\begin{aligned} &\mathbf{X}^{(t)} = \mathbf{A}\mathbf{X}^{(t-1)} + \mathbf{c} + \boldsymbol{\varepsilon}^{(t)}, \\ &\boldsymbol{\varepsilon}^{(t)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}), \\ &\boldsymbol{\Sigma}_{ij} = \sigma^2 \exp(-\phi \| h(i) - h(j) \|), \end{aligned} \tag{4}$$

where $h(i) \in \mathbb{R}^2$ denotes the location of sensor i and $\| \cdot \|$ is Euclidean norm. In this case, the model we wish to learn happens to be similar to (1), the model for suppression. However, these models serve different roles and in general can have different forms; our framework does not assume them to be similar. At the lower level, this model also differs from (1) in two ways. First, $\mathbf{X}^{(t)}$ here represents readings of all sensors in the network at time t (as opposed to just those inside one cluster as in (1)), and dimensions of other matrices and vectors are scaled similarly. Second, we impose on $\boldsymbol{\Sigma}$ an exponential structure, one of the most widely-used isotropic covariance functions for modeling spatial correlation [17].

For a total of m sensors and n timesteps, we write all sensor readings in an $n \times m$ matrix $\mathbf{X}$, where the i-th row $\mathbf{X}^{(i)}$ corresponds to all readings at time i, and the j-th column corresponds to all readings from sensor j. Due to suppression and failures, parts of the matrix are missing; we denote them by $\mathbf{W}$. The remaining entries are received by the base station. We denote by $\mathbf{V}$ the received entries as well as the derived constraints (cf. Section 6). $\mathbf{V}$ represents our knowledge about $\mathbf{X}$.

Model Priors Prior distributions of the model parameters can encode specific domain knowledge about the sensor data. In the absence of such information, non-informative priors can be used. For CS2, we use a (conjugate) inverse Gamma prior for the σ^2 parameter, and flat priors for $\mathbf{A}$ and $\mathbf{c}$. For ϕ, we apply the commonly used uniform prior after discretization.

Sampling In order to draw from the posterior joint distribution, we use the well-known Gibbs sampling technique [16]. The basic idea is to draw a sequence of samples from the distribution of each variable, conditioned on all others. After a sufficient number of iterations, the chain converges to the target distribution.

To seed the iterations, we obtain an initial sample for each model parameter from its prior, and we produce initial values for $\mathbf{W}$ by linear programming subject to all constraints in $\mathbf{V}$. In iteration k, we sample $\mathbf{W}^{[k]}, \mathbf{A}^{[k]}, \mathbf{c}^{[k]}, (\sigma^2)^{[k]}, \phi^{[k]}$ in turn. First, we sample $\mathbf{W}^{[k]}$ one row at a time. Recall that some components in $\mathbf{X}^{(t)}$ may be missing, hence in $\mathbf{W}$. We partition $\mathbf{X}^{(t)}$ into two parts: the missing part, $\mathbf{W}^{(t)}$, and the known part, $\mathbf{V}^{(t)}$. It can be easily shown that the conditional distribution of $\mathbf{W}^{(t)}$ given the latest samples of all others is MVN (details omitted). Standard techniques can be used to sample $\mathbf{W}^{(t)}$ subject to the linear constraints. Next $\mathbf{A}$ and $\mathbf{c}$ can be sampled using the well-known Metropolis-Hastings algorithm, where the Metropolis step for each component can be independent random walks. Lastly we update $(\sigma^2)^{[k]}$ by sampling from an inverse Gamma distribution and $\phi^{[k]}$ from a discrete distribution (mathematical derivation is simple and omitted).

8 Experimental Evaluation

All experiments use a two-tier suppression hierarchy.

Datasets For evaluation, we use both synthetic and real-world datasets. We generate the synthetic dataset by a VAR(1) model (whose parameters values are not known to our suppression and inference algorithms). The redwood dataset [2] is a collection of microclimate data collected by sensors deployed on redwood trees in Sonoma, California. We use the humidity data in this dataset, and assume VAR(1) in inference. The humidity dataset is representative of many types of environmental data, which follow a general trend and have strong spatial correlation. Of course, suppression will not work if data is dominated by random noise. However, we have found many types of data amenable to suppression (e.g. light under foliage). In this paper, we report results on the humidity dataset. Note that the synthetic dataset allows us to compare inference results with the *true* model parameters used for data generation; the redwood dataset is less suitable for this purpose because the *true* model is unknown and too complex to be precisely fit. Thus there is no ground truth to which inference results can be compared. However, it is still useful for testing communication reduction and missing data recovery.

Approaches Compared We have implemented the following approaches: **Temporal**, an implementation of BaySail [4], where each node reports directly to the base station using value-based temporal suppression with timestamp repetition for redundancy; **Ken**, based on the disjoint-cliques model in [3]; and **CS2**. Both Ken and CS2 use a VAR(1) model for suppression between a cluster head and the base station. However, Ken does not use any suppression for intra-cluster communication.

Depending on the redundancy scheme used, we have two variants of CS2: **CS2-R**, with the baseline redundancy scheme of Section 5.1 based on simple repetition, and **CS2-C**, with the coding-based improvement of Section 5.2. Note that the original Ken framework has no special handling of failures. To make meaningful comparisons, we also consider an enhancement to Ken, **Ken-ACK(r)**, which uses ARQ for redundancy and allows at most r retransmissions for each message. To improve energy efficiency, we let the cluster head broadcast a single ACK message with a bitmap indicating which children's transmissions were received, instead of acknowledging each child separately. The receiver always assumes suppression if no message is received.[3] We have also added Bayesian inference to Ken and Ken-ACK(r).

Metrics of Evaluation We evaluate two aspects of data collection schemes: energy cost (both communication and computation) and inference quality (recovering missing data and learning model parameters).

We estimate the total energy cost by $N_{\text{comm}} + \delta N_{\text{comp}}$, where N_{comp} is the number of CPU operations performed by spatiotemporal suppression algorithms (cost of simple value-based temporal suppression is negligible). δ is a ratio of the cost of a CPU operation (e.g., scalar multiplication) to that of transmitting 1 byte over one hop in the network. We use $\delta = \frac{1}{450}$ in our experiments, based on Tmote Sky nodes' energy profile. The communication cost N_{comm} is further modeled as $N_c + \lambda N_b$, where N_c is the total number of bytes transmitted within clusters and N_b is the total number of bytes transmitted between cluster heads and the base station. λ is the per-byte cost ratio of head-to-base communication to child-to-head communication. In a multihop network, a cluster head is typically closer to its child nodes than to the base station, so the cost of head-to-base communication is higher ($\lambda > 1$). Our calculation of N_c and N_b includes per-message overheads, as they can be significant for small messages. Based on the IEEE802.15.4 standard, implemented by the CC2420 radio stack and TinyOS, the packet overhead is 10 bytes. Sensor readings and timestamps take 2 and 0.5 bytes each, respectively.

Inference quality is measured by *Mean Squared Error*, defined as the average of the squares of the "errors" (difference between samples and the true value). It incorporates both the bias and the variance of the estimation.

Communication Reduction (Figure 6) We put failures aside for now and first demonstrate the "bottom-line" communication efficiency of cascaded suppression compared with Temporal and Ken. For this experiment, we assume reliable communication (no failures). No redundancy is added for any of the three approaches. The head-to-base/child-to-head communication cost ratio λ is set to 2. Given a global suppression threshold ϵ, CS2 distributes it evenly to the two tiers of suppression edges. For Ken and CS2, the quadratic subset selection algorithm is used. We take 1000 timesteps of data from the redwood dataset and measure the average communication cost per node per timestep. We vary the global suppression threshold as well as the cluster size, and plot the results in Figure 6.

The first observation is that CS2 significantly outperforms the other two schemes in all cases. Temporal obviously has the highest communication cost, and it stays virtually the same as the cluster size increases. On the other hand, CS2 and Ken are able to exploit the increasing spatial correlation to save communication. Ken is outperformed by CS2 because it incurs full intra-cluster communication costs. Also, the curve for Ken is relatively flat as ϵ changes, probably because spatial suppression is less sensitive to suppression threshold changes. In contrast, a temporal component in the suppression scheme helps save more communication when ϵ increases, as in the cases of Temporal and CS2.

Finally, we note that as clusters get larger in Ken and CS2, the amount of additional communication savings diminishes, because distant sensors have weaker correlation. Furthermore, larger clusters increase computation costs, as we have discussed in Section 4 and will demonstrate by experiments later in this section. Therefore, large clusters are practically not attractive.

To recap, cascaded suppression is more communication-efficient than previously proposed suppression methods, and performs well across varying cluster sizes and suppression thresholds.

Inference Quality (Figure 7) Having seen how cascaded suppression reduces communication, we next assess the effectiveness of CS2 in recovering missing data and learning model parameters. We use both the redwood dataset and the synthetic dataset, each containing data collected by 4 nodes over 200 timesteps. This time, we consider failures. We let each message fail with probability 30%, a rate we have experienced in real-world deployments. We test CS2-R, with all redundancy levels set to 2 and the suppression thresholds for both tiers set to 0.5. The Gibbs sampling algorithm is run for 10000 iterations; samples generated in the "burn-in" period are discarded, leaving ≈ 8000 samples for each variable. Figure 7 shows examples of sample distribution.

The top plots show the sample distribution for two randomly chosen missing sensor readings in the redwood dataset, and the bottom for parameters ϕ and σ^2 in the VAR(1) model generating the synthetic dataset. All samples are roughly centered around the true values, which means our Bayesian inference algorithm is effective. Note that ϕ's sample distribution has a long right tail, which is not visible at the scale shown.

Choice of Subset Selection Algorithms (Figure 8) Recall the two greedy algorithms in Section 4 for selecting a subset of readings to transmit when total suppression is not possible. We now show their energy profile and impact on inference quality in the context of CS2-R. Again, the failure rate is set to 30%. The head-to-base redundancy level is set to 1 and all others to 2.

We generate 200 timesteps of synthetic sensor data for a network with 18 nodes, placed on a 3×6 grid and grouped into 3 equally-sized clusters. The moderate cluster size is motivated by the discussion earlier in the context of Figure 6. The overall suppression threshold is equally distributed to the two tiers. This setup, which we call **Syn18**, is used in all remaining experiments.

We have already hinted on the tradeoff between energy cost and inference quality from the perspective of

3. As argued in Section 5, however, ARQ still cannot guarantee reliable transmission, so this assumption may be questionable in practice. Nonetheless, for the purpose for comparison with our techniques, we shall give Ken-ACK(r) the advantage of ignoring this problem.

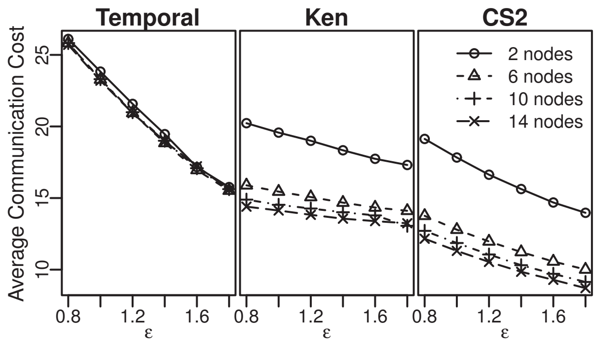


Fig. 6. Communication cost.

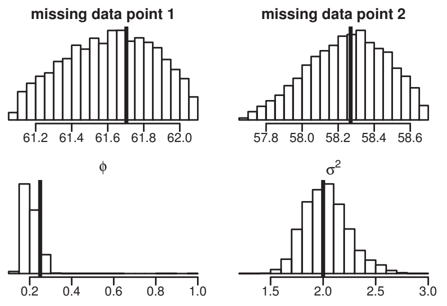


Fig. 7. Distribution of samples for missing data and model parameters (bold vertical line: true value).

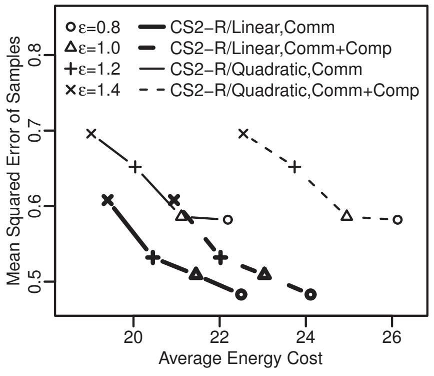


Fig. 8. Subset selection algorithms.

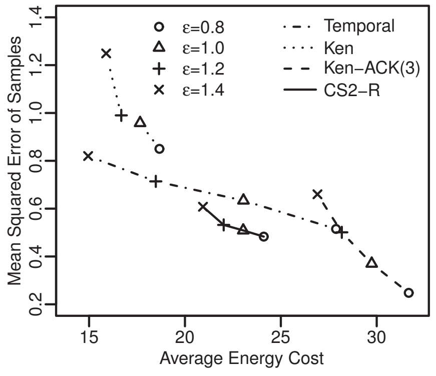


Fig. 9. Data collection approaches.

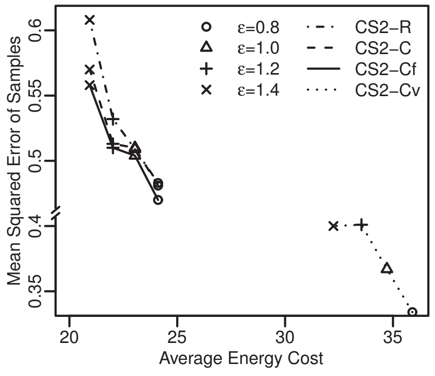


Fig. 10. Coding schemes.

suppression algorithms. To improve inference quality for a given suppression scheme, one typically has to use more energy. Figure 8 shows this tradeoff by varying the suppression threshold and plotting the inference quality against energy cost for each setting. To further illustrate the overhead of computation, we plot not only the cost of communication but also the total cost.

Comparing the quadratic and linear algorithms in Figure 8, we see that the quadratic algorithm has a much higher cost because of its computational complexity, which is expected from the analysis in Section 4. We also see that computation can account for a significant fraction of the total cost, confirming the importance of bounding algorithmic complexity and cluster sizes.

Another interesting observation is that the linear algorithm has better inference quality. As evidenced by its higher communication costs, the linear algorithm transmits slightly more readings than the quadratic algorithm, and these additional readings help improve inference quality. Furthermore, the extra communication costs are more than enough compensated for by the lower computation costs. In the experiments that follow, we only use the linear algorithm for subset selection.

Holistic Comparison of Data Collection Approaches (Figure 9) We now compare how various suppression-based data collection approaches trade off between energy cost and inference quality. We consider CS2-R with $r_R = 2$, and $r_S = 2$ (for child-to-head suppression edges) or 1 (for head-to-base suppression edges), and three other approaches: Temporal (with redundancy level set to 2), vanilla Ken, and Ken-ACK(3). The message failure rate is 30% and the head-to-base/child-to-head communication cost ratio $\lambda = 2$. We test all approaches on the Syn18 setup with different suppression thresholds.

Figure 9 shows that vanilla Ken has the highest inference error, although its energy cost is also low because it employs no redundancy for coping with failures. Once we allow retransmissions, as in Ken-ACK(3), energy cost drastically increases despite our broadcast optimization. Although Ken-ACK(3)'s inference quality seems to improve steadily, this improvement is subject to the caveat discussed in Footnote 3. Our cascaded suppression approach, CS2-R, is able to hit some "sweet spots" in the space of tradeoffs. With comparable energy costs, CS2-R is able to achieve better inference quality than Temporal. With less energy, CS2-R is able to achieve similar inference quality as Ken-ACK(3) without being subject to the same caveat.

Impact of Coding Schemes (Figure 10) As discussed in Section 5.2, we have flexibility in applying our coding-based technique to improve the redundancy scheme. We now evaluate the effect of different coding schemes by considering variants of CS2. We take CS2-R from the previous experiment (Figure 9) as the baseline for comparison. The first coding-based scheme we consider is **CS2-C**, which uses the encoder in Figure 3(a) for type-S redundancy for head-to-base suppression edges (child-to-head suppression edges are handled as in CS2-R). Another scheme we consider, **CS2-Cf**, is obtained by replacing the encoder of CS2-C with the one with feedback in Figure 3(b). These two schemes encode only transmission bit vectors, timestamps, and indicators ($\surd$, $\times$ $\perp$, or ?). We evaluate one more scheme, **CS2-Cv**, which uses the same encoder as CS2-C but encodes the actual values transmitted instead of just transmission bit vectors. Using the same settings as in the previous experiment, we plot the results in Figure 10.

From the figure, we see that CS2-Cv achieves much smaller errors than others, but incurs higher communication costs at the same time. This tradeoff is expected because CS2-Cv encodes more information using more bytes. The other approaches have identical communication costs (because of the redundancy settings) but different inference qualities. As expected, CS2-C and CS2-Cf results in smaller inference error than CS2-R, due to the advantage of coding-based redundancy over simple repetition-based based redundancy. CS2-Cf performs even better than CS2-C, perhaps because the feedback structure of the encoder provides more protection against failures. This experiment shows that a careful choice of the encoder can lead to additional benefit; also, selecting different types of information to code (e.g., CS2-Cv) can further extend the range of quality-cost tradeoff.

9 Conclusion

Continuous data collection is a basic task in many applications of wireless sensor networks. To reduce the energy cost of communication, we have proposed cascaded suppression. We have shown that cascaded spatiotemporal suppression is more flexible and effective than previously proposed suppression schemes. More importantly, our comprehensive solution tackles the problems of handling transient message failures, interpreting missing data, and learning from data.

Failure handling is particularly challenging for cascaded suppression, because nodes can act on inaccurate information and in turn affect other nodes. We resolved this problem by logging and forwarding essential information to the base station to allow reconstruction of history and interpretation of data. We have further applied convolutional coding techniques to the transmission of such information, using a novel decoding algorithm that does not make traditional assumptions such as the existence of good failure models. This feature, together with the fact that the correctness of suppression does not depend on the correctness of its model, make our solution especially suited for data collection tasks in unfamiliar and unpredictable environments.

References

[1] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *WSNA*, 2002.

[2] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay *et al.*, "A macroscope in the redwoods," in *SenSys*, 2005.

[3] D. Chu, A. Deshpande, J. Hellerstein, and W. Hong, "Approximate data collection in sensor networks using probabilistic models," in *ICDE*, 2006.

[4] A. Silberstein, A. Gelfand, K. Munagala, G. Puggioni, and J. Yang, "Making sense of suppressions and failures in sensor data: a Bayesian approach," in *VLDB*, 2007.

[5] R. Johannesson and K. S. Zigangirov, *Fundamentals of Convolutional Coding*. Wiley-IEEE Press, 1999.

[6] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks," in *OSDI*, 2002.

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

[8] D. Tulone and S. Madden, "PAQ: Time series forecasting for approximate query answering in sensor networks," in *EWSN*, 2006.

[9] D. Slepian and J. Wolf, "Noiseless coding of correlated information sources," *IEEE Transactions on Information Theory*, vol. 19, no. 4, pp. 471–480, 1973.

[10] S. Pradhan, J. Kusuma, and K. Ramchandran, "Distributed compression in a dense microsensor network," *Signal Processing Magazine, IEEE*, vol. 19, no. 2, pp. 51–60, 2002.

[11] Z. Xiong, A. Liveris, and S. Cheng, "Distributed source coding for sensor networks," *Signal Processing Magazine, IEEE*, vol. 21, no. 5, pp. 80–94, 2004.

[12] A. Guitton, N. Trigoni, and S. Helmer, "Fault-tolerant compression algorithms for delay-sensitive sensor networks with unreliable links," in *DCOSS*, 2008.

[13] H. Yang and C.-W. Chung, "An effective and efficient method for handling transmission failures in sensor networks," in *DASFAA*, 2009.

[14] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

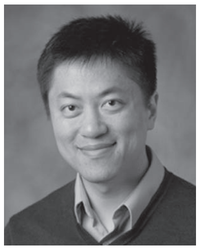
[15] R. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Transactions on Information Theory*, vol. 9, no. 2, pp. 64–74, 1963.

[16] A. Gelfand and A. Smith, "Sampling-based approaches to calculating marginal densities," *JASA*, vol. 85, no. 410, pp. 398–409, 1990.

[17] S. Banerjee, B. Carlin, and A. Gelfand, *Hierarchical modeling and analysis for spatial data*. Chapman & Hall, 2004.

Yi Zhang received his B.E. from Tsinghua University in 2006 and is currently a Ph.D. student of Computer Science at Duke University. His research interests include uncertain data processing in sensor networks, statistical data management and data-intensive computing.

Kristian Lum received her Ph.D. from the Department of Statistical Science at Duke University in Durham, North Carolina in 2010. Prior to that, she completed her B.A. in Mathematics and Statistics from Rice University in Houston, Texas in 2006. She has worked on projects in varied applications areas, including wireless sensor networks and population size estimation. Her main research focus is Bayesian spatial statistics, particularly relating to quantile regression.



Jun Yang received his B.A. from University of California at Berkeley in 1995, and his Ph.D. from Stanford University in 2001, both in Computer Science. He is currently an Associate Professor of Computer Science at Duke University. He is broadly interested in research on data management and data-intensive computing. He is a recipient of the National Science Foundation CAREER Award and the IBM Faculty Award.