A Robust Architecture for Distributed Inference in Sensor Networks

Mark Paskin
Stanford University
Computer Science Department
Stanford, California 94305
Email: mark@paskin.org

Carlos Guestrin and Jim McFadden
Carnegie Mellon University
School of Computer Science
Pittsburgh, Pennsylvania 15213
Emails: {guestrin,jmcfadde+}@cs.cmu.edu

***Abstract*— Many inference problems that arise in sensor networks require the computation of a global conclusion that is consistent with local information known to each node. A large class of these problems—including probabilistic inference, regression, and control problems—can be solved by message passing on a data structure called a *junction tree*. In this paper, we present a distributed architecture for solving these problems that is robust to unreliable communication and node failures. In this architecture, the nodes of the sensor network assemble themselves into a junction tree and exchange messages between neighbors to solve the inference problem efficiently and exactly. A key part of the architecture is an efficient distributed algorithm for optimizing the choice of junction tree to minimize the communication and computation required by inference. We present experimental results from a prototype implementation on a 97-node Mica2 mote network, as well as simulation results for three applications: distributed sensor calibration, optimal control, and sensor field modeling. These experiments demonstrate that our distributed architecture can solve many important inference problems exactly, efficiently, and robustly.**

I. Introduction

Sensor networks consist of nodes that can measure characteristics of their local environment, perform computations, and communicate with each other over a wireless network. In recent years, advancements in hardware and low-level software have led to viable, multi-hundred node sensor networks that can instrument unstructured environments at an unprecedented scale. For example, the Mica2 "mote" can measure temperature, humidity, pressure, visible and infrared light, sound, magnetic fields, and acceleration. The most popular application of sensor networks to date has been environmental monitoring. In these deployments the sensor data is downloaded from the network for later analysis [1] or the network aggregates the measurements using simple local operations that compute, for example, averages, maxima, or histograms [2], [3].

More advanced applications, such as tracking and actuation, require sensor networks that can solve significantly more complex problems like sensor fusion, data modeling, prediction, and optimal control. Solving these inference problems requires combining all of the nodes' local measurements to generate a globally consistent view of the environment, or in the case of actuation, coherent controls to change it. For example, a node with a temperature sensor can measure only the temperature at its location; if the node's sensor is biased, it is impossible to infer the true temperature from the measurement. However, by combining this local information with the measurements of the other sensors, the network can solve a global inference problem that automatically calibrates the temperature sensors at all nodes.

Most existing inference algorithms for sensor networks focus on solving specific tasks such as computing contour levels of sensor values [4], distributed sensor calibration [5], or target tracking [6]. In this paper, we present the first general architecture for inference in sensor networks that can solve a wide range of problems including probabilistic inference problems (e.g., sensor calibration and target tracking), regression (e.g., data modeling and contour finding), and optimization (e.g., actuator control, decision-making and pattern classification). At the core of the architecture is a powerful data structure called a *junction tree*, which allows all of these inference problems to be solved by simple message passing algorithms [7].

Recently, there have been some proposals to use existing centralized inference algorithms in sensor networks, e.g., belief propagation [8], [9] and particle filtering [10]. However, these inference approaches are not as general as ours, and more importantly, they do not fully address the practical issues that arise in real deployments: communication over wireless networks is unreliable due to noise and packet collisions; the wireless network topology changes over time; and, nodes can fail for a number of reasons, often because their batteries die. To address these challenges, we have found that it is insufficient to implement existing algorithms on the sensor network architecture; fundamentally new algorithms are required.

To address these robustness issues we propose a novel architecture consisting of three layers: *spanning tree formation*, *junction tree formation*, and *message passing*. The nodes of the sensor network first organize themselves into a spanning tree so that neighbors have high-quality wireless connections. Using pairwise communication between neighbors in this tree, the nodes compute the information necessary to transform the spanning tree into a junction tree for the inference problem. In addition, these two algorithms jointly optimize the junction tree to minimize the computation and communication required by inference. Finally, the inference problem is solved exactly via message passing on the junction tree. These three algorithms quickly recover from communication and node failures by reacting to changes in each others' states. We demonstrate the viability of our architecture with experiments on a sensor network of 97 Mica2 motes, and we illustrate its generality with simulation experiments on three different inference tasks using data from a real sensor network deployment. An extended version of this paper presents additional background, details, and experiments [11]. For further detail on the application of our architecture to probabilistic inference and regression problems, see [12], [13].

A. Inference problems in sensor networks

Our architecture is useful for solving a wide variety of inference problems that arise in sensor networks. Below we describe three examples to give a sense of the range of problems addressed. See [7] for more examples.

Perhaps the most intuitive example is **optimal control**, where the nodes can control their environment to achieve some end. Consider a greenhouse deployment where nodes actuate the blinds to achieve specific desired light levels at different locations. The light level measured by each node will depend on the states of nearby blinds, and nearby nodes may have conflicting desires. To achieve the setting of the blinds that are best for all of the nodes, we can specify for each node a *reward function* that specifies its local utility for a setting of the blinds given its current light measurement. For example, if the light location for node i depends upon blinds 1 and 2 then $Q_i(a_1, a_2; \overline{m}_i)$ is a local reward function that depends upon its current light measurement $\overline{m}_i$ and the controls a_1, a_2 applied to nearby blinds (e.g., open or close). Because each blind may affect the light level at several nodes, these reward functions may represent conflicting interests. To balance the nodes' competing desires, we can select the controls that maximize the sum of all nodes' reward functions:

$$\mathbf{a}^* = \underset{\mathbf{a}}{\operatorname{argmax}} \sum_{i=1}^{n} Q_i(\mathbf{a}; \overline{m}_i) \tag{1}$$

If solved by enumerating the possible control settings, this optimization problem requires exponential time. However, by exploiting locality structure—each reward function depends upon a small set of controls—the problem can often be solved efficiently. Consider a simple example with 3 binary controls, $\{a_1, a_2, a_3\}$, where the reward function is $Q(a_1, a_2) + Q(a_2, a_3)$. Using distributivity, we can rewrite the maximization problem as:

$$\max_{a_1,a_2,a_3} Q(a_1, a_2) + Q(a_2, a_3) = \max_{a_1,a_2} Q(a_1, a_2) + \max_{a_3} Q(a_2, a_3).$$

In this simple example, we can decrease the number of operations from 16 to 12. More generally, we can exploit distributivity to "push" the $\max$ over each control variable past all terms of the sum that do not depend upon its value, potentially obtaining exponential decrease in complexity [14].

Probabilistic inference is a powerful tool for solving problems where we must reason with partial or noisy information [15]. These problems often arise in sensor networks, where the sensor measurements give an incomplete view of the environment. The general task is to compute the posterior distributions of some desired quantities given a probabilistic model of the environment and a set of observed measurements. Many challenging problems can be solved using probabilistic inference; as an example, consider the *distributed sensor calibration* task [5]. In this problem our nodes obtain measurements of some field (e.g., temperature), and these measurements are corrupted by unknown, independent biases. The task is to automatically remove these biases by exploiting the correlation between the measurements obtained by nearby nodes. To accomplish this with probabilistic inference, we require a prior probabilistic model of the temperature field, the measurements, and the biases; given the local correlation structure, a natural choice for this prior is a *graphical model*, whose distribution is a product of local terms [15]. Given this model, we can compute the posterior distribution of the true temperatures by instantiating the observed measurements, multiplying together the terms of the model, and marginalizing out nuisance variables. These posterior temperature estimates automatically "calibrate" the sensors and also account for measurement noise. As in the control problem, we can exploit distributivity to push these marginalizations past the multiplications to obtain an efficient inference algorithm [15]. The application of our architecture to probabilistic inference problems is considered in depth in [12].

Another important task that arises in sensor networks is **regression**, or function fitting. Many current sensor network deployments are used for data gathering: all of the network's measurements are

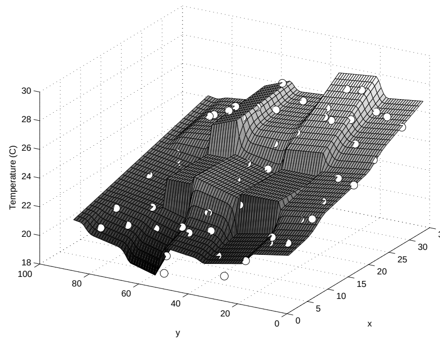


Fig. 1. The temperature measurements from a sensor network deployed in the Intel Berkeley lab and a regressed function.

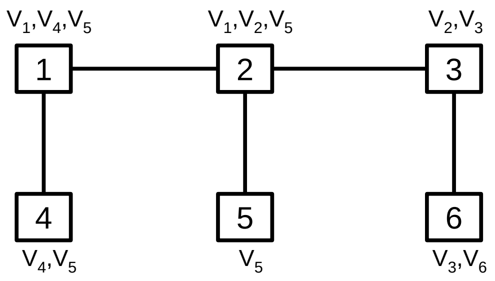


Fig. 2. Example of a junction tree.

uploaded to a central location. This is wasteful when the measurements at nearby locations are correlated (as in the temperature measurements of the previous example). Regression is a powerful and general framework for maintaining the structure of the sensor field while significantly decreasing the communication required to access it [13]. In *linear regression*, the sensor field is modeled by a weighted combination of basis functions: $\hat{f}(x, y, t) \triangleq \sum_{j=1}^{k} w_j b_j(x, y, t)$ represents an approximation to the value of the sensor field at location (x, y) at time t. The $b_j(x, y, t)$ are basis functions which are chosen in advance, and the weights w_j are optimized to minimize the sum squared error between the observed measurements and the model $\hat{f}$. The optimal weights constitute a low-dimensional summarization of the original data that can be communicated off the network with significantly less cost. In the general case, computing the optimal weights requires solving a dense linear system. *Kernel linear regression* is a specialization of this technique where each basis function has bounded support (i.e., a local region of influence), and the optimal weights are the solution to a *sparse* linear system. Figure 1 shows the result of fitting such a function to sensor network temperature data. As shown in [13], these regression problems also have significant locality structure and a distributive property that can be exploited to yield an efficient inference algorithm.

B. Message passing on junction trees

The inference problems above may seem very different, but they have a common algebraic structure [11]. Each problem requires us to first combine local pieces of information about a set of variables to obtain a global model, and then summarize this model to a subset of variables; for example, the control problem is specified by a set of local reward functions which are combined (by addition) to form a global reward function and then summarized (by maximization) to determine the optimal actions for a subset of the control variables. Because these problems share this essential structure, they all can be solved by algorithms that pass messages on a *junction tree* [7]. Below we describe this important data structure as well as the structure of the message passing algorithms. The references above describe the message passing operations used in each type of inference problem.

The problems described above each have a set of **variables** $V_1, \ldots, V_n$, which are the objects of inference: in probabilistic inference, these are the random variables of the model; in regression, they are the optimal weights; and in control, they are the control variables. A **clique tree** is an undirected tree where each node i is associated with a subset of the variables $\mathbf{C}_i$, called its **clique**. In the clique tree example in Figure 2, we have that $\mathbf{C}_2 = \{V_1, V_2, V_5\}$. In the message passing inference algorithms, each node begins with local information about (a subset of) the variables in its clique; by passing messages along the edges of the tree, the nodes obtain "summaries" of the relevant information that is stored by other nodes. Informally, the message that node i sends to node j is computed by combining node i's local information with the information it obtains in messages from neighbors other than j, and then "summarizing away" information about variables that are not in $\mathbf{C}_j$. In Figure 2, when sending a message to node 3, node 2 combines information from nodes 1 and 5 with its local information, and then summarizes away variables V_1 and V_5 that are not present in C_3. (For example, in the control problem node i adds together local reward functions it obtains from neighbors other than j, and maximizes out all control variables that are not in node j's clique.) These messages may be scheduled synchronously so each message is computed only once, or they may be sent asynchronously so that they converge to the correct values. Once a node's incoming messages are available, it can combine them with its local information to obtain the globally correct result for its clique of variables. For example, in the control problem, a node can compute optimal settings for its clique of control variables; in the regression problem, a node can compute an optimal estimate of the sensor field in a local neighborhood; and, in the probabilistic inference problem, a node can compute the posterior distribution of its clique of random variables given the measurements made by all nodes in the network.

To guarantee the correctness of these message passing algorithms, the clique tree must satisfy a structural constraint called the **running intersection property**:

If a variable is in cliques $\mathbf{C}_i$ and $\mathbf{C}_j$, then it must also be in all cliques on the (unique) path between nodes i and j.

If this property holds, the tree is called a **junction tree**. Note that Figure 2 is a junction tree, e.g., V_5 appears in C_4 and C_5, thus it also appears in C_1 and C_2 to satisfy the running intersection property. The running intersection property guarantees that the nodes reasoning about any variable V_i form a subtree. Intuitively, this structure guarantees that by pairwise exchanges of information, all nodes reasoning about V_i can reach consensus. Because this structure holds for all variables simultaneously, complete global consistency is also reached by this local communication.

C. Overview of the architecture

There are two types of information that are relevant to solving inference problems in sensor networks: prior information (such as the reward functions of a control problem or the basis functions of a regression problem), and measurements that are obtained by the sensors. For simplicity we assume the prior information has been distributed to the nodes of the network (perhaps before deployment, or via dissemination techniques), and each node has obtained its sensor readings; thus, each node i begins with local information about some subset of the variables $\mathbf{D}_i$ (which may overlap). These **local variables** are not known to the other nodes.

If we were to now organize the nodes of the sensor network into an undirected tree, then we would have a distributed data structure that is almost a junction tree; all that would be missing are the cliques associated with each node. This hints at a three-layer architecture for distributed inference: (1) the **spanning tree layer** allows each node to select a set of neighbors with good communication links such that the nodes are organized in a spanning tree; (2) the **junction tree layer** allows the nodes to compute their cliques to transform the spanning tree into a junction tree that is "embedded" in the network; and (3) the **inference layer** allows the nodes to asynchronously pass the inference messages over the edges of the junction tree, each node eventually converging to the correct result of inference for its clique of variables. The next three sections describe these layers.

II. Spanning tree formation

The goal of the spanning tree layer is for each node to choose a set of neighbors so that the nodes form a spanning tree where adjacent nodes have high-quality communication links. In wireless sensor networks, this problem is very challenging: link qualities are asymmetric and change over time; and, nodes must discover new neighbors and estimate their associated link qualities, as well as detect when neighbors disappear. Fortunately, spanning trees are well studied in distributed systems and sensor networks (e.g., for multi-hop routing [16]).

Our application has unique requirements, so we found it necessary to develop a distributed spanning tree algorithm specifically for our architecture. In addition to being correct and robust to failure, we require a spanning tree algorithm that is **stable**—the tree must remain fixed whenever possible—and **flexible**—we would like to choose between a wide variety of different trees. These properties are important for routing, but not crucial: the main goal of routing is to move packets through the network. In our setting, the spanning tree defines a "logical architecture" for our inference algorithm; thus, the spanning tree algorithm must be as stable as possible so that the inference algorithm can make progress. The spanning tree also determines the computation and communication required to solve the inference problem, so we must be able to flexibly choose between different spanning trees to minimize the cost of inference.

To achieve these goals our spanning tree algorithm builds upon existing algorithms. As in the IEEE 802.1d protocol, the nodes of the network elect the node with the lowest identifier as the root, and each node chooses a parent node that offers a path to the root. To ensure stability under changing network conditions, the nodes compute robust link quality estimates using exponentially-weighted moving averages [16] and use them to select edges whose bidirectional link quality is consistently good. To flexibly choose between multiple trees, we have developed descendant test strategies that give each node a larger choice of valid parents. For details on the spanning tree algorithm see [11].

III. Junction tree formation

Recall that each node i in the sensor network starts with local information about a set of variables $\mathbf{D}_i$. Once a spanning tree has been built, the nodes have formed a distributed data structure similar to a junction tree: a tree where each node has local information about a subset of the variables (see Figure 3). To make this into a junction tree, we must also specify the clique $\mathbf{C}_i$ for each node i of the network. These cliques must satisfy two properties: each node's clique must include its local variables ($\mathbf{C}_i \supseteq \mathbf{D}_i$ for all nodes i); and, we must have the running intersection property: if two cliques $\mathbf{C}_i$ and $\mathbf{C}_j$ have the same variable V, then all nodes on the unique path between them must also carry V.

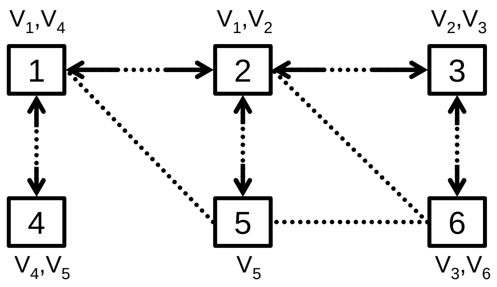


Fig. 3. Example of the initial spanning tree in a six node network; the dotted lines indicate high-reliability links, the links used in the spanning tree are shown with arrows. Next to each node i is the domain $\mathbf{D}_i$ of its local factor. Note that the running intersection is not satisfied; $\mathbf{D}_4$ and $\mathbf{D}_5$ include V_5, but $\mathbf{D}_1$ and $\mathbf{D}_2$ do not.

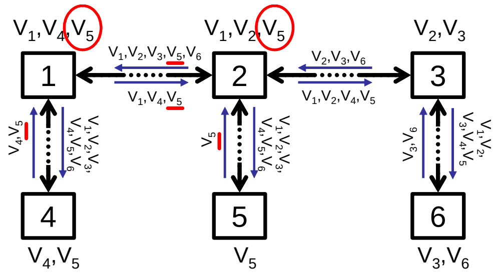


Fig. 4. The reachable variables messages for Figure 3. Each node i is now labeled with its clique $\mathbf{C}_i$. The reachable variables message $\mathbf{R}_{32} = \{V_2, V_3, V_6\}$ is obtained by the union of $\mathbf{R}_{63} = \{V_3, V_6\}$ with the local variables for node 3, $\mathbf{D}_3 = \{V_2, V_3\}$. The circled variables were added to satisfy the running intersection property, e.g., V_5 is included in $\mathbf{C}_2$ because it appears in $\mathbf{R}_{12}$ and $\mathbf{R}_{52}$, as shown by the underlined variables in the messages.

Below we present a robust, distributed algorithm that passes messages between neighbors in the spanning tree in order to compute the unique set of minimal cliques that satisfy these two properties. Because the spanning tree topology determines the cliques of the junction tree, we also present a robust, distributed algorithm for optimizing the spanning tree to induce cliques that minimize the communication and computation required by inference.

A. Ensuring the running intersection

We begin by presenting the algorithm under the assumptions that there is a stable, valid spanning tree and that communication between neighbors is reliable. Then we generalize it to the case where these assumptions do not hold, and we describe optimizations that minimize communication.

1) Message passing algorithm: Each node learns its clique using a message passing algorithm in which it sends a message to and receives a message from each neighbor. Let i be a node and j be a neighbor of i; the **variables reachable to** j **from** i, $\mathbf{R}_{ij}$, are:

$$\mathbf{R}_{ij} \triangleq \mathbf{D}_i \cup \bigcup_{k \in \text{nbr}(i) \setminus j} \mathbf{R}_{ki}, \tag{2}$$

These messages are defined recursively; the base case is a message from a leaf node, which is simply that node's local variables. An interior node i computes $\mathbf{R}_{ij}$ by collecting the variables that can be reached through each neighbor but j and adding its local variables $\mathbf{D}_i$; then it sends $\mathbf{R}_{ij}$ as a message to j. Figure 4 shows the reachable variables messages for the example of Figure 3.

If a node receives two reachable variable messages that both include some variable V, then it knows that it must also carry V to satisfy the running intersection property. Formally, node i computes its clique $\mathbf{C}_i$ using

$$\mathbf{C}_i \triangleq \mathbf{D}_i \cup \bigcup_{\substack{j,k \in \text{nbr}(i) \\ j \neq k}} \mathbf{R}_{ji} \cap \mathbf{R}_{ki}. \tag{3}$$

For example, in Figure 4, node 2 receives two reachable variables messages that contain V_5, and so its clique must include V_5, as shown. Using the reachable variables messages, a node i can also compute its **separator** $\mathbf{S}_{ij} \triangleq \mathbf{C}_i \cap \mathbf{C}_j$ with a neighbor j, via $\mathbf{S}_{ij} = \mathbf{C}_i \cap \mathbf{R}_{ji}$; this is the set of variables common to nodes i and j, and it determines the size of the inference messages they exchange.

To make this message passing algorithm asynchronous, each node initializes its incoming reachable variables messages to be empty. Each time node i receives a new reachable variables message from a neighbor j, it recomputes its reachable variables messages to all neighbors but j, and transmits them if they have changed from their previous values; in addition, it recomputes its clique and separators. This algorithm is guaranteed to converge to the unique, minimal set of cliques that preserve the running intersection property for the underlying spanning tree.

2) Robust, distributed implementation: In the presentation above we assumed reliable communication between neighbors in the spanning tree. While this is not true at the physical layer, it can be implemented at the transport layer using message acknowledgments—by hypothesis, the spanning tree consists of high-quality wireless links. We also assumed that the reachable variables messages were transmitted after the spanning tree algorithm had run to completion. The algorithm cannot be implemented in this way, though, because in a sensor network, there is no way to determine when a distributed algorithm has completed. For example, a node can never rule out the possibility that a new node will later join the network.

Our algorithms therefore run concurrently on each node, responding to changes in each others' states. When the spanning tree layer on a node adds or removes a neighbor, the junction tree layer is informed and reacts by updating its reachable variables messages. If node i obtains a new neighbor j, then $\mathbf{R}_{ij}$ is computed and sent to j; if j is removed from i's neighbor set then for all other neighbors k, $\mathbf{R}_{ik}$ is recomputed and retransmitted (if it has changed from its previous value). This tight interaction between the layers permits the junction tree to reorganize quickly when changing link qualities, interference, or node failures cause the spanning tree to change.

3) Minimizing communication: This junction tree algorithm is the only part of our architecture where nodes must reason about "global" aspects of the inference problem. In general, the space complexity of the reachable variables messages is linear in the total number of variables; for example, if j is a leaf in the spanning tree, then $\mathbf{R}_{ij}$ must include all variables (except possibly $\mathbf{D}_j$). For large problems, then, it is important to choose a compact encoding of the reachable variables, e.g., bit vectors or sets of integer intervals, to minimize communication cost.

As we have described the algorithm above, $\mathbf{R}_{jk}$ is retransmitted whenever it changes, which can happen when j receives a new reachable variables message from another neighbor. A great deal of communication can be saved if instead of sending the new value of $\mathbf{R}_{jk}$, node j sends a "patch" that allows node k to compute the new value from the old one. In the full version of the paper we describe an optimized protocol in which nodes transmit an *add set* and a *drop set* to compactly communicate updates to the reachable variables messages; we also describe how problem-specific structure can be exploited to reduce communication [11].

B. Optimizing the junction tree

The algorithm above transforms the spanning tree into a junction tree by computing the unique set of minimal cliques that satisfy the running intersection property. Note that different spanning trees can give rise to junction trees with different clique and separator sizes; for example, if in Figure 3 node 5 had chosen to connect to node 1 instead of node 2, the node 2's clique would not need to include the variable V_5. The size of a node's clique determines the amount of computation it must perform, and the separator sizes determine the amount of communication required by neighbors in the tree. These facts motivate a *tree optimization algorithm* that chooses a spanning tree that gives rise to a junction tree with small cliques and separators.

The input to this algorithm is a cost function that decomposes over the cliques and separators of the junction tree. For example, to minimize the computation and communication required to solve the inference problem, we may choose $\alpha_i(\mathbf{C}_i)$ to be the (energy) cost of the inference computations required by node i if its clique is $\mathbf{C}_i$, and $\beta_{ij}(\mathbf{S}_{ij})$ to be the communication cost paid by node i to send an inference message to node j, if their separator is $\mathbf{S}_{ij}$. The total cost is:

$$\sum_{i=1}^{N} \left[\alpha_i(\mathbf{C}_i) + \sum_{j \in \mathrm{nbr}(i)} \beta_{ij}(\mathbf{S}_{ij}) \right] \tag{4}$$

These cost functions can take into account the problem-specific costs of the inference algorithm as well as network characteristics such as link qualities and (perhaps heterogeneous) processor speeds.

Finding the spanning tree that minimizes this cost function is NP-hard (by a simple reduction from centralized junction tree optimization [15]), but we can define an efficient distributed algorithm for greedy local search through the space of spanning trees. First we use the spanning tree algorithm to build up a good spanning tree using link quality information only. Then the tree optimization algorithm repeatedly reduces the cost of inference by performing legal edge swaps; for example, in Figure 3 node 5 can swap its edge to 1 for an edge to 2 or an edge to 6.

Nodes learn about a legal edge swap, and the change to the global cost (Eq. (4)) that would occur if it was implemented, using a distributed dynamic programming algorithm. By starting an **evaluation broadcast** along one of its spanning tree edges, a node can learn about alternatives for the edge and their relative costs. For example, in Figure 3, suppose node 5 sends to its neighbor 2 a message $\mathsf{EVAL}(5, 2)$, meaning "find legal alternatives for our edge $5 \leftrightarrow 2$." Node 2 then propagates $\mathsf{EVAL}(5, 2)$ to its other neighbors, nodes 1 and 3. When node 1 receives the message, it sees that the originator, 5, is a potential neighbor, and propagates the message back to 5 *outside the spanning tree*. When node 5 receives the $\mathsf{EVAL}(5, 2)$ message from node 1, it learns of a legal swap: it can trade its edge to 2 for an edge to 1. Similarly, node 3 propagates the request to node 6, which then propagates it to node 5 outside of the spanning tree; in this way node 5 learns $5 \leftrightarrow 6$ is another alternative for $5 \leftrightarrow 2$.

In general, swapping spanning tree edges has non-local effects on the cliques and separators of the induced junction tree, so a node cannot assess the relative cost of an edge swap locally. However, the relative cost can be assessed efficiently by an extension of the evaluation broadcast scheme described above. The key idea is that if the edge $5 \leftrightarrow 2$ were swapped for the edge $5 \leftrightarrow 1$, *only the reachable variables messages (and cliques) on the cycle* $5 \leftrightarrow 2 \leftrightarrow 1 \leftrightarrow 5$ *would change.* This is a direct consequence of the definition of the reachable variables messages in Eq. (2). Similarly, if the edge $5 \leftrightarrow 2$ were swapped for the edge $5 \leftrightarrow 6$, only the reachable variables messages on the cycle $5 \leftrightarrow 2 \leftrightarrow 3 \leftrightarrow 6 \leftrightarrow 5$ would change. Therefore, to evaluate the relative cost of an edge swap, only the nodes on the **swap cycle**, i.e., the cycle closed by the new edge, must participate in the computation. As the evaluation messages propagate around the swap cycle, each node adds in its local contribution to the cost estimate; to do this, it computes the reachable variables messages, clique, and separators it would have if the swap were implemented, and evaluates the change to the cost in Eq. (4). The evaluation protocol is described in detail in [11].

The node that initiated the evaluation broadcast collects responses and performs the edge swap that minimizes the cost of the tree. If two nodes undertake edge swaps at the same time and their swap cycles overlap, then the resulting change in cost may be different than the individual cost estimates would indicate. To coordinate these updates, evaluation broadcasts are used only when snooping the broadcast channel indicates no other evaluations are in progress. When there are no conflicting edge swaps, this distributed algorithm will converge to a junction tree that is a local minimum of the cost function.

As we have described it, the communication pattern of the tree optimization algorithm is expensive: when node i starts an evaluation broadcast via a neighbor j, the evaluation messages are propagated to all nodes on the j side of the $i \leftrightarrow j$ edge. Fortunately, for typical cost functions it is possible to prove that once the running value of change in cost becomes positive, it can never decrease as the evaluation messages propagate. Because we are not interested in swaps that increase the tree cost, we can halt propagation of the evaluation messages whenever the running cost becomes positive. Another method to reduce the communication cost is to use a hop count limit to limit the local search.

IV. Inference by message passing

The top layer in our architecture is a robust, distributed implementation of the message passing algorithm for solving the inference problem. The details of these algorithms vary across different problems (e.g., probabilistic inference [12] or regression [13]), but the structure is the same: the message that node i sends to node j depends upon node i's local information, the messages it receives from all neighbors but j, and the separator $\mathbf{S}_{ij} \triangleq \mathbf{C}_i \cap \mathbf{C}_j$. Thus, whenever any of these quantities changes, the message is recomputed and retransmitted. For example, if a node i receives an updated message from a neighbor, it recomputes and retransmits its messages to all other neighbors; if the junction tree layer signals that the separator $\mathbf{S}_{ij}$ to node j has changed, then node i recomputes and retransmits its message to node j. If the spanning tree eventually stabilizes, then the junction tree will also stabilize; in this case these rules guarantee that the inference messages will eventually converge to the correct values, and the nodes will stop passing inference messages.

In some problems, it is possible to make intelligent decisions about when retransmitting a message is not worth the communication cost. For example, if node j has transmitted a message to node k and it then receives a new message from another neighbor, it often happens that the updated message it would send to k is not that different from the previous value. In some cases it is possible to obtain error bounds associated with suppressing message updates; this can be an effective way to trade communication cost for approximation error [11].

In our architecture we have achieved robustness with a tight interaction between the layers: each layer responds to changes in the states of the other two to react to changing network conditions. But now we have reached the top of our algorithm stack, and we must consider how an application will use the results of inference when it cannot be sure that the inference algorithm has run to completion.

Certainly the solution to this problem will be application specific, but it seems clear that in general it is useful for the inference algorithm to guarantee that at any point during its execution, each node's **partial result**—i.e., the quantity which is computed when not all of the final versions of the messages have arrived—is useful. Some inference algorithms naturally have this property: in the regression algorithm the partial result represents the optimal estimate given only the measurements obtained by nodes in communication range [13]. Other inference algorithms do not have this property: the partial results of the traditional algorithm for probabilistic inference can be arbitrarily far from the correct results. To make these algorithms useful for inference in sensor networks, extra work is necessary; for example, see [12] for a new message passing algorithm for probabilistic inference that resolves the problem.

V. Experimental results

In this section, we evaluate our architecture and algorithms on a real Mica2 sensor network and on a realistic simulator. Here we present a brief summary of our findings; see the full version of the paper for more detail [11].

A. *Mica2 mote implementation*

Our first set of experiments test the spanning tree and junction tree layers on a real sensor network. We implemented these two layers in TinyOS and deployed the architecture on a sensor network at the Intel Berkeley Research Lab; see Figure 5(a). The network has 97 Mica2 motes, each of which is equipped with a 433Mhz radio, a 8MHz microprocessor, 4KB of RAM, and 512KB of flash memory. Each mote is connected to a power supply and ethernet, which was used only for instrumentation. For our evaluation of these two layers we used a kernel regression inference problem [13] with 28 variables (basis function coefficients) and 84 nodes.

Figure 5(b) shows the communication properties of the junction tree layer; it plots the total number of bytes of reachable variables messages (represented using a bit encoding) sent during each second the algorithm runs. There are no messages sent in the beginning because the spanning tree layer is estimating link qualities. Once the spanning tree layer begins establishing links, the number of reachable variables messages increases. Soon after, the running intersection property is satisfied and communication ceases. We ran the algorithm for hours at a time and found that the tree was remarkably stable: on average it ran for 30 consecutive minutes without sending a single reachable variables message.

We also tested the robustness of the spanning and junction trees to both node and communication failures. We simulated node and link failures by signaling individual motes (using the testbed ethernet connections) to either die or ignore messages from a given neighbor. Figure 5(c) shows that our architecture is very robust, recovering rapidly from failures. Soon after a failure, communication increases as messages are sent to restore the running intersection property, but the tree stabilizes rapidly. Note that the communication cost of repairing an existing tree is much lower than building the initial tree.

To quantify the communication saved when repairing a broken junction tree, we ran 30 experiments where groups of random nodes were killed. (In this experiment, we used 78 of the nodes.) Figure 5(d) shows the average number of bytes of reachable variables messages necessary to build an initial junction tree and also to recover from failures of one to five randomly selected nodes. We found that on average, a node sends only 13 reachable variables messages to build the initial junction tree; this indicates that building the initial junction tree requires a modest amount of communication. In addition, repairing the junction tree after failures requires even less communication; e.g., recovering from a simultaneous five-node failure requires about one third the communication of building the junction tree from scratch.

For simplicity, we did not implement the complete tree optimization algorithm in § III-B on the motes. Instead, we used a simple neighbor selection heuristic that chooses, among the neighbors that have above average link quality, the one whose initial clique has highest intersection with this node's initial clique. We found that using this heuristic decreased communication cost in the inference layer by a factor of 33% over considering link quality alone.

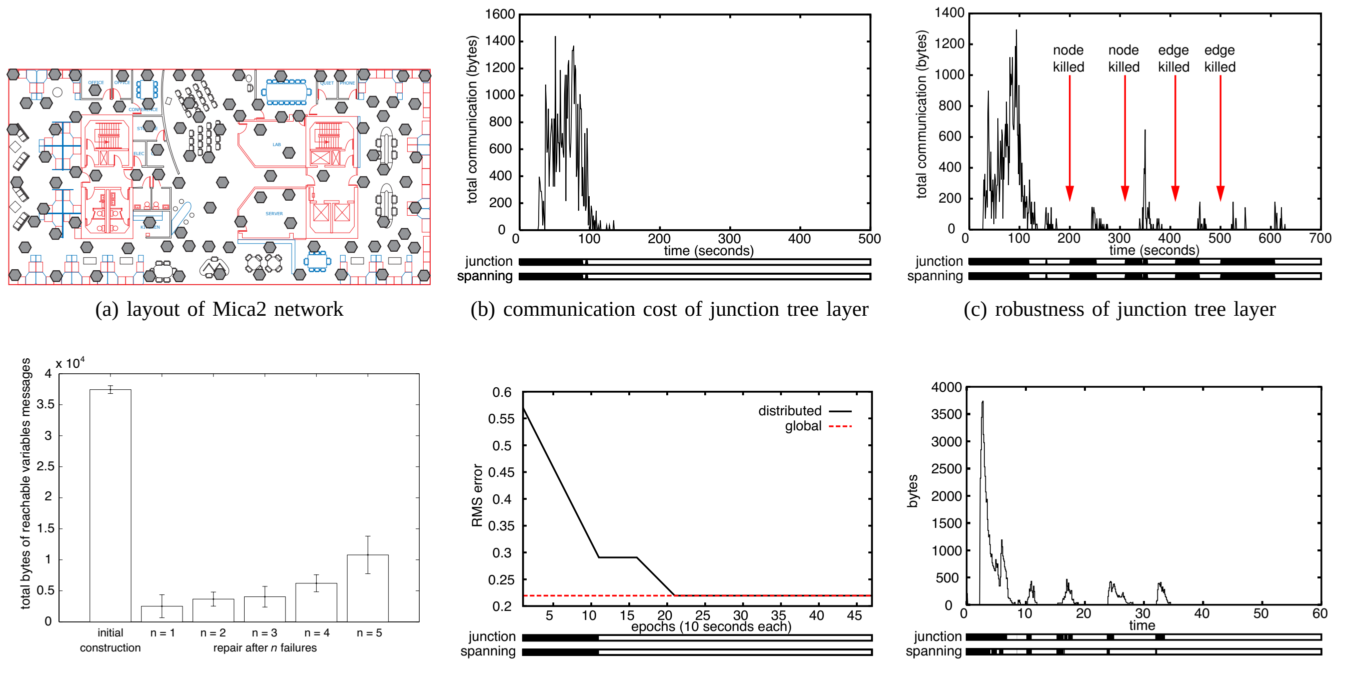
In addition to the spanning tree and junction tree layers described thus far, we have an initial Mica2 implementation of the message passing layer for the kernel regression problem described in §I-A. We have performed preliminary experiments using a sensor network with 15 Mica2 motes on a regression problem with 2 kernels and 4 spatial and temporal basis functions per kernel. In these experiments (see Figure 5(e)), the nodes converged to the same regression coefficients as the optimal offline solution after only 20 epochs. Despite the fact that Mica2s only have software fixed-point arithmetic capabilities, the matrix operations required by regression were stable and precise. The messages in this layer used reliable communication with acknowledgments, requiring between 3 and 7 36-byte packets per message. The main limiting factor in the Mica2s is the small amount of RAM (4KB), which did not allow us to hold all of the necessary matrices in memory at once. We addressed this problem by using block matrix operations that page unused parts of the matrices to flash memory.

B. *Simulation experiments*

To further test our architecture and algorithms, we designed a network simulator based on data and link qualities from a different deployment of 53 Mica2 motes. To verify that our simulations are realistic, we simulated the communication cost experiment described above. The simulated results in Figure 5(f) are qualitatively similar to the real results in Figure 5(b); in fact, the real network seems more stable than the simulated one. This gives us some confidence that the simulation results will also hold on a real network.

We ran another experiment to test the distributed tree optimization algorithm. We chose our communication cost function to be proportional to the expected number of transmitted bytes necessary to successfully communicate the inference messages (for the calibration problem described below), taking into account retransmissions. The piecewise constant curve in Figure 6(a) represents the current cost of the spanning tree when one exists; the horizontal line represents a hypothesized optimum: it is the cost of the best tree we were able to find using centralized optimization techniques. Note that the initial spanning tree, which is selected using only link quality information, is significantly more expensive than the hypothesized optimum, but that the distributed optimization algorithm eventually finds trees whose cost is within a factor of two.

The next set of experiments were performed on the distributed sensor calibration problem described in §I-A. Using the temperature data from the real network, we learned a Gaussian graphical model over the true temperatures, biases, and temperature measurements. To set up our distributed sensor calibration task we created an artificially biased set of measurements by sampling a bias for each node and adding these biases to a held-out test set of measurements. The inference task is for the nodes to estimate these biases from the corrupted observations, using the probabilistic model. Each node uses probabilistic inference to compute its posterior mean bias estimate,



(a) layout of Mica2 network
(b) communication cost of junction tree layer
(c) robustness of junction tree layer
(d) communication required to recover from failure
(e) preliminary regression results
(f) communication cost (simulation)

Fig. 5. Experimental results. In figures (b), (c), (e), and (f), the x-axis is time; the bottom bar shows when a valid spanning tree has been constructed, and the top bar shows when the running intersection property has been enforced.

and the error metric we report is the root mean squared error (RMS) from these estimates to the biases we sampled.

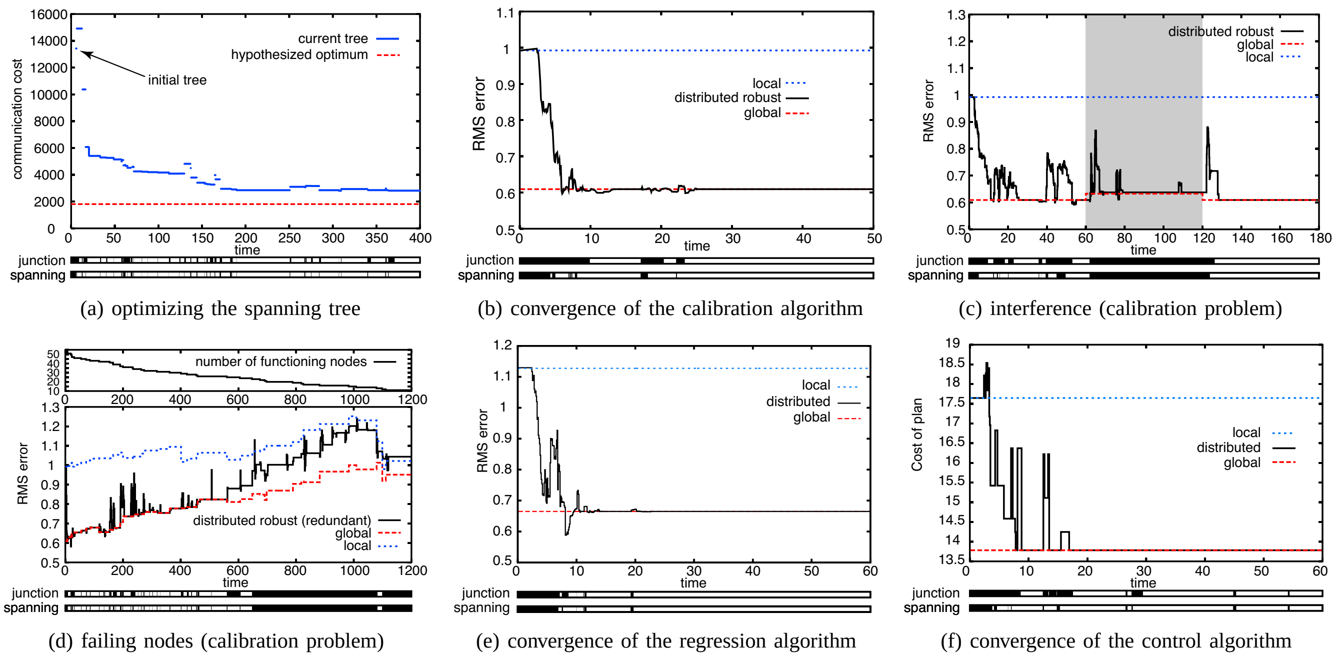
Figure 6(b) visualizes a trace of the inference architecture when the robust message passing algorithm of [12] is used to solve the probabilistic inference problem. The main panel of Figure 6(b) plots the RMS error of three inference algorithms. The line marked *global* refers to centralized inference using all of the measurements. In this case, the posterior mean bias estimates of global inference have 0.61 RMS error. Thus, by solving the global inference problem the nodes can automatically eliminate 39% of the bias. The line marked *local* refers to centralized local inference, where each node's posterior is computed using only its measurement. Local inference performs about as well as predicting zero bias, achieving a 0.99 RMS error; this is expected, since correlated measurements from different nodes are required for automatic calibration. The third curve, *distributed robust*, refers to our architecture. This plot graphically demonstrates the key properties of the algorithm: before any messages have been passed, the partial results coincide with the estimates given by local inference; at convergence, the estimates coincide with those of centralized global inference; and, before all messages have been passed, the estimates are informative approximations. Looking closely, we can see that before the junction tree is valid, and even before a complete spanning tree is constructed, the estimates of the robust message passing algorithm quickly approach the exact solution.

To test the algorithms' robustness to long-term communication failure, we ran the same experiment, but this time we introduced a period where interference causes the network to be segmented into two parts. In Figure 6(c) we can see that the algorithm converges before and after the inference period, but that interference prevents a (complete) spanning tree from being formed. In spite of this, the robust message passing algorithm converges to an excellent approximation: each half of the network forms its own junction tree and performs inference with the available information.

We also tested the architecture's performance under simulated node failures. Figure 6(d) shows the results of this experiment. As each node dies, its measurement is lost, so the inference problem to be solved is changing over time; this explains the changing error values for global and local inference. Notice that the network can form a junction tree and solve the inference problem exactly past 500 seconds, when only 26 of the original 53 nodes are still functioning.

Our next experiment evaluates our architecture on the regression task described in §I-A. Using the distributed regression formulation described in [13], we defined a regression problem on the temperature data with 22 basis functions. In our regression task, each node uses its local estimate of the optimal model parameters to predict the measurement of its five nearest neighbors, along with its own measurement. Figure 6(e) shows the resulting root mean squared error for this task. As with the calibration case, this graph shows three curves: the *local* curve corresponds to each node using its own measurement to predict its neighbors' measurements; the *global* curve corresponds to fitting the regression parameters offline, and using the resulting model for prediction; the *distributed* line uses our architecture and the distributed regression messages so that each node locally predicts its neighbors' values using its current estimates of the basis function coefficients. As with the calibration case, we see that the results obtained by our distributed algorithm quickly converge to those obtained by the optimal offline solution.

Our third and final inference problem is an instance of the control problem described in §I-A. We defined an actuation problem where 16 blinds can be moved to change the light conditions; each blind is controlled by a specific node of the network. Each actuating node has five possible controls which raise and lower the blinds by varying amounts. Each node of the network has a desired light value that is 40 lux greater than its current value. The goal is to find positions for all the blinds that minimize the mean squared deviation from the desired light values. Our results, shown in Figure 6(f), again compare three methods: in the *local* curve each actuating node chooses the blind setting that best fits its own desires; the *global* curve corresponds to



(a) optimizing the spanning tree (b) convergence of the calibration algorithm (c) interference (calibration problem)

(d) failing nodes (calibration problem) (e) convergence of the regression algorithm (f) convergence of the control algorithm

Fig. 6. Experimental results from simulation.

the optimal solution obtained offline; the *distributed* curve uses our architecture to optimize the setting in a distributed fashion, where each actuating node chooses the control setting that it currently views as the best global solution. As with calibration and regression, we see that the control strategy obtained by our distributed algorithm quickly converges to that obtained by the optimal offline solution.

VI. Conclusions

We presented the first general and robust architecture for inference in sensor networks that can solve a wide range of inference problems including probabilistic inference, regression, and optimization. In particular, we have presented distributed algorithms that can construct stable junction trees, even in the presence of communication and node failures; we have also presented distributed algorithms to optimize this junction tree to minimize the cost of inference, and to solve the inference problem. We demonstrated the viability of the architecture in a real sensor network deployment, and we demonstrated its generality with three applications—distributed sensor calibration, sensor field modeling, and optimal control—using a realistic sensor network simulation. Our results demonstrate that the architecture is robust to communication and node failures, and that in all three applications the inference algorithm quickly converges to the correct answer.

An important feature of our architecture is that it does not rely on a network layer that provides multi-hop routing (which is difficult or impossible in sensor networks). This is due in part to the communication pattern of our algorithms: only neighbors in the tree must communicate with each other. Another reason is that our architecture tightly couples the application and networking layers so that both network-related and application-specific information can be used to minimize the communication and computation required by inference. We expect that this tight coupling between the application and networking layers is useful for other types of in-network computation.

General architectures that address a range of sensor network applications (and the robustness issues of real systems) will significantly increase the usefulness of sensor network technology. We believe that the work presented herein provides a solid step toward this goal.

Acknowledgements: We gratefully thank Phil Buonadonna, David Gay, Joe Hellerstein, Wei Hong, Sam Madden, Romain Thibaux, and Alec Woo for helpful discussions and assistance with the sensor network deployments.

References

[1] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," Tech. Rep. IRB-TR-02-006, Intel Research, 2002.

[2] J. Hellerstein, W. Hong, S. Madden, and K. Stanek, "Beyond average: towards sophisticated sensing with queries," in *IPSN*, 2003.

[3] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann, "Impact of network density on data aggregation in wireless sensor networks," in *Int'l. Conf. on Distributed Comp. Systems (ICDCS)*, 2002.

[4] R. Nowak and U. Mitra, "Boundary estimation in sensor networks: Theory and methods," in *IPSN*, 2003.

[5] V. Byckovskiy, S. Megerian, D. Estrin, and M. Potkonjak, "A collaborative approach to in-place sensor calibration," in *IPSN*, 2003.

[6] F. Zhao, J. Liu, J. Liu, L. Guibas, and J. Reich, "Collaborative signal and information processing: An information directed approach," *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1199–1209, 2003.

[7] S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE Trans. on Information Theory*, vol. 46, pp. 325–343, 2000.

[8] C. Crick and A. Pfeffer, "Loopy belief propagation as a basis for communication in sensor networks," in *Uncertainty in Artificial Intelligence (UAI)*, 2003.

[9] K. Plarre and P. R. Kumar, "Extended message passing algorithm for inference in loopy Gaussian graphical models," *Ad Hoc Networks*, vol. 2, pp. 153–169, 2004.

[10] M. Coates, "Distributed particle filtering for sensor networks," in *Information Processing in Sensor Networks (IPSN)*, 2004.

[11] M. Paskin and C. Guestrin, "A robust architecture for distributed inference in sensor networks," Tech. Rep. IRB-TR-03-039, Intel Research, 2004.

[12] M. Paskin and C. Guestrin, "Robust probabilistic inference in distributed systems," in *Uncertainty in Artif. Intelligence (UAI)*, 2004.

[13] C. Guestrin, R. Thibaux, P. Bodik, M. Paskin, and S. Madden, "Distributed regression: an efficient framework for modeling sensor network data," in *Information Processing in Sensor Networks*, 2004.

[14] C. Guestrin, D. Koller, and R. Parr, "Multiagent planning with factored MDPs," in *Neural Information Processing Systems*, 2001.

[15] R. Cowell, P. Dawid, S. Lauritzen, and D. Spiegelhalter, *Probabilistic Networks and Expert Systems*, Springer, New York, NY, 1999.

[16] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in *Int'l. Conf. on Sensor Systems (SENSYS)*, 2003.