# Supervised Learning in Sensor Networks: New Approaches with Routing, Reliability Optimizations

Yong Wang, Margaret Martonosi, and Li-Shiuan Peh
Princeton University, Princeton, NJ 08544
{yongwang, mrm, peh}@princeton.edu

*Abstract*— **Routing in sensor networks maintains information on neighbor states and potentially many other factors in order to make informed decisions. Challenges arise both in (a) performing accurate and adaptive information discovery and (b) processing/analyzing the gathered data to extract useful features and correlations. To address such challenges, this paper explores using supervised learning techniques to make informed decisions in the context of wireless sensor networks.**

**In consideration of the unique characteristics of sensor networks, our approach consists of two phases: an offline learning phase and an online classification phase. We use two case studies to demonstrate the effectiveness of our approach. In the first, we present MetricMap, a metric-based routing protocol that derives link quality using our classifiers when the traditional ETX-based approach fails. In the second, we present SHARP, an extension to the PSFQ protocol, which uses knowledge gathered in the training phase to control its caching policy for saving constrained storage space. Evaluation is performed on a 30-node real-world testbed and a multihop sensor network in our lab. Our results show that MetricMap can achieve up to 300% improvement in data delivery rate for a high data-rate application, without compromising other performance metrics; SHARP can reduce the memory footprint of PSFQ by 46.4% with a modest increase of 4.7% in fetch miss rate.**

## I. Introduction

Many critical applications in wireless sensor networks rely fundamentally on fast, efficient, and reliable data delivery. In order to overcome the inherent unreliability of sensor network communication links, communication protocols increasingly employ intricate and situation-aware adaptations to identify good routes and to determine resource-efficient methods for handling data.

The difficulties in situation-aware adaptations are two-fold. First, some adaptation techniques are hard-wired heuristics based on observations of a few stylized types of network problems and their solutions. The more problems one envisions, the more complicated the protocol becomes in trying to adapt around them. Second, environmental factors interact in such complex ways that it can be difficult to identify correlations and crisply define the problem scenarios to protect against.

In this paper, we explore using machine learning techniques to improve situation-awareness in order to optimize sensor network routing. Machine learning is an effective and practical technique for discovering relations and extracting knowledge in cases where the mathematical model of the problem may be too expensive to get, or not available at all. Supervised learning is a particular case when the inputs and outputs are both given in the training phase. For example, inputs might include node-level and network-level metrics, such as buffer occupancies, channel load assessments, packet received signal strength, etc. Output may be the expected number of transmissions over the link where the packet is received. Essentially, we aim to use machine learning to *automatically* discover correlations between readily-available features and the quantity of interest. Supervised learning is an effective learning technique in solving this type of problem.

We manage the resource constraints of sensor networks by employing machine learning in a two-phase method: an offline training phase followed by an online classification. Offloading the training task from sensor nodes reduces the processing, communication, and energy requirements of the node during deployment. The resulting classifiers used online are both strikingly lightweight and effective. For the case studies examined in this paper, our learning framework results in prediction accuracies of 80% or more, with false positive rates between 4.1% and 11.3%, and with essentially no compute overhead after deployment.

We present two case studies of supervised learning for routing and reliability optimizations. In the first case study, we present MetricMap, a metric-based data collection protocol atop MintRoute that predicts link quality using our classifiers in a highly congested network. In the second case study, we present SHARP, a situation-aware reliable transport protocol atop PSFQ that uses knowledge gathered from the training phase to control its caching policy in order to save constrained storage space, while ensuring reliability.

The primary contributions of this work are summarized as follows:

1) We present a general framework that uses supervised learning to extract information automatically within sensor networks. Our method is automatic, which is advantageous over heuristics-based methods whose effectiveness may depend on the context where they are developed and evaluated.
2) We cast link quality estimation and packet caching prediction as classification problems, which permits the use of simple, yet effective existing learning algorithms. Decision tree learners and rule learners represent such algorithms. We believe a large range of applications can benefit from this approach.
3) We present an evaluation of our approach using implementations in TinyOS on real-world sensor networks.

Our results show that MetricMap can improve over existing approaches by up to a factor of 3 when traffic rate is higher than 2pps (packet per second) and SHARP can save 47% storage over PSFQ with a modest increase in fetch miss rate.

The rest of this paper is organized as follows. In Section II, we introduce the background knowledge related to the discussion of the work. Next, we describe the details of our learning framework in Section III. In Sections IV and V, we present two cases studies and results of our prototype implementation on real-world sensor networks. Related work is discussed in Section VI and the last section summarizes the main results and outlines future work.

## II. Background

### A. Link Quality Estimation

Wireless sensor networks are very different from wired networks in that the link quality fluctuates greatly as a consequence of interference and propagation dynamics. Therefore, developing efficient routing in sensor networks requires the establishment of high quality paths, which in turn entails accurate knowledge of link quality. In this section, we briefly review the mechanisms behind existing link quality estimation methods, including both software-based and hardware-based ones. We also explain how they fail to function when the traffic rate becomes high. This motivates our work on new approaches based on machine learning.

**Software-based estimation.** ETX [1], also proposed in MintRoute [2], is defined as the expected number of transmissions (including retransmissions) for a successful end-to-end data forwarding and hop-by-hop acknowledgment.

We focus here on the snooping-based method adopted by MintRoute.[1] It defines link quality as

$$etx(l) = \frac{1}{p_f(l) \times p_r(l)}$$

with $p_f(l)$ the forward probability of link $l$ and $p_r(l)$ its reverse probability. $p_f(l)$ is calculated using the ratio of the number of data packets received to the total number of data packets transmitted over $l$. $p_r(l)$ is calculated as $p_f(\bar{l})$ with $\bar{l}$ the reverse link of $l$. The route metric of a $n$-hop path $p$ is then calculated as $ETX(p) = \sum_{i=1}^{n} etx(l_i)$, the total expected number of (re)transmissions along the path.

However, in many high data-rate applications [4], [5], snooping-based link quality estimation works poorly. For example, consider the structure monitoring application described in [4]. Due to structural vibration damping effects, a very high data sampling rate is required, which is estimated to be at least 200Hz. This leads to a data rate as high as 9.6kbps per node with each node sampling 16-bit in three spatial dimensions. Even with in-network processing techniques, such as data aggregation, compression and coding, the expected traffic is still very challenging for current systems to cope with. Several real-world testbed evaluations of high data-rate applications have been reported [6], which demonstrate the severeness of the *data funneling effect* during high traffic load.

[1]The difference between the two approaches is studied in [3].

**Hardware-based estimation.** The link quality indication (LQI) metric characterizes the strength and/or quality of a received packet. It is introduced in the 802.15.4 standard [7] and is provided by CC2420, the radio in MicaZ and Telos motes. LQI measures the incoming modulation of each successfully received packet. The resulting integer ranges from 0x00 to 0xff, indicating the lowest and highest quality signals detectable by the receiver (between -100dBm and 0dBm). LQI values are uniformly distributed between these two limits. Prior work [8] on the Telos motes shows that average LQI closely tracks average success rate of packet transmissions across several links.

In this paper, we use LQI to label link quality in each training sample. We do not use LQI directly for link quality estimation during routing because LQI is only available with each successfully received packet. In a congested network, the number of received packets is small and LQI is not a reliable measure of link quality. Our approach can avoid this problem by tracking features that are available all the time. If some feature is missed, it will use other features to infer the situation based on the knowledge obtained from training.

### B. Hop-by-hop Caching for Reliability

In this section, we briefly review reliable data transport, an important area of research in sensor networks, which will be used as our second case study.

PSFQ [9] (Pump Slowly, Fetch Quickly) is a reliable block transport protocol designed to deliver an ordered block of packets from one location (sink) to individual sensors or a set of sensors, such as in the case of network reprogramming or complex query injection. Clearly, losses in these applications are not tolerable.

In PSFQ, the source **pumps** data in sequence to the destination. Whenever an out-of-order arrival is detected, a **fetch** operation is started to perform local recovery by asking (NACK) for the lost packets from its immediate parent. PSFQ uses hop-by-hop caching to localize loss recovery to only one hop. This requires the intermediate nodes to passively cache all data packets. However, passive caching is not always desirable. In situations where network conditions are good, there is no need to cache packets that will never be fetched. For example, Paek et al. report in [4] that retransmission rate is as low as 3-7% in a similar scenario. Therefore, the caching necessity depends on situations. If we can predict which packet needs to be cached, better resource efficiency will be achieved. The penalty of such a caching scheme is to retransmit packets that are not cached locally. In that case, the node issuing the fetch request has to go beyond one hop to recover its lost packets. This is a tradeoff between caching accuracy and (re)fetch overhead.

Once we cast the prediction problem of "fetch" or "no fetch" into a classification problem, the prediction of possible fetches can be solved using supervised learning.

## C. Supervised Learning Overview

The goal of supervised learning is to predict the value of an outcome measure based on a number of input measures [10]. The outcome measure could be numerical or categorical. Learning is performed on a set of training samples. Each sample ($x_i$,$y_i$) consists of a feature vector $x_i$ and a corresponding class label or numerical value $y_i$. The feature vector contains measurable features of the system under consideration. If the outcome is categorical, the learning becomes a classification problem. Training a classifier usually involves finding a mapping from feature vectors to output labels so that the overall classification error is minimized on the training samples. A good learner should accurately predict new samples not in the training set. Therefore, given a classification problem, we need to decide (a) what features to measure and (b) what learning algorithm to use to maximize the learning accuracy.

In this paper, we evaluated two classifiers — *decision tree learners* and *rule learners*. There exist other, more sophisticated, methods of classification, including support vector machines, Bayesian networks, and ensemble methods. Any such learner can be used as the classifier for our technique. However, our results show that decision tree learners and rule learners produce remarkably good accuracy for our case studies and many times they achieve the highest accuracy among all algorithms studied. Also, we prefer learners that produce human-readable outputs and both decision tree learners and rule learners are good for this purpose.

**Decision tree learners.** Decision tree learners are widely used in solving classification problems with classifiers represented as trees. They take a "divide-and-conquer" approach and recursively divide attributes at each internal node in the tree based on information they possess. Leaf nodes represent classification decisions. Pruning methods are used to prevent overfitting of training data. Although decision tree learners are not always the most competitive learners in terms of accuracy, they are computationally efficient and the results produced can be easily converted to human-readable formats.

**Rule learners.** Rule learners are used for learning IF-THEN rules. Rule learners work on training samples with similar input/output pairs as decision tree learners. However, since the rule-sets learned are disjoint to each other, they usually produce far fewer rules than decision tree learners on the same training set, with a comparable accuracy. This makes it preferable in scenarios where the size of classifiers matters.

**Learning overhead.** Since wireless sensor networks are constrained in node processing time, energy usage, and memory footprint, we need to also consider such overhead, in addition to learning accuracy. We focus on the overhead of online classification and feature collection since training is conducted offline in our learning framework, usually on a resource-rich backend server.

To utilize the output of a decision tree learner, we need to translate it into IF-THEN rules. As the number of produced rules is as many as the number of leaf nodes in the tree, a large tree with hundreds of leaves results in hundreds of rules to be hand-coded into the online classifier. Instead, we use rule learners to implement the online classifier due to their small footprints and acceptable accuracies.

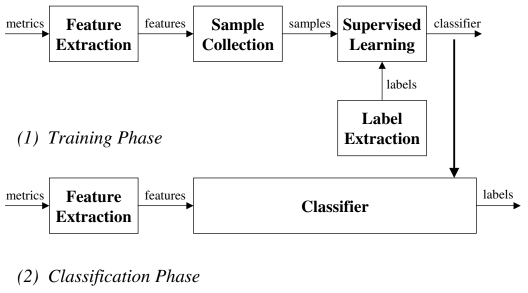


Fig. 1. Overview of learning steps.

**Learning cost.** Given a classifier and an instance, there are four possible outcomes. If the instance is positive and it is classified as positive, it is counted as a *true positive* (TP). On the other hand, if the instance is negative and it is classified as positive, it is counted as a *false positive* (FP). TP rate is defined as the ratio of positives correctly classified to the total positives. FP rate is defined as the ratio of negatives incorrectly classified to the total negatives.

It is crucial for a real-world application to consider FP rate since it represents the *cost* of learning. Usually we want a high TP rate (high benefits) and a low FP rate (low costs).

## III. Learning Step-by-Step

In this section, we review the steps of our proposed learning framework using link quality classification as an example. Figure 1 presents a high level overview of the steps involved, with the four key steps listed as follows:

1) Select the features to be used in training and classification;
2) Instrument every node in the network to collect these features and their corresponding labels and periodically send them back to the sink;
3) Use the labeled data to perform training at the sink node;
4) Instrument MintRoute to use the classifier for differentiating between high quality and low quality links at runtime. The algorithm is depicted in Section IV.

In what follows, we describe the first three steps with specific reference to a collection routing application. Since the last step is closely related to the application, we discuss application instrumentation in Section IV.

### A. Step 1: Feature Extraction and Output Labeling

The first step in supervised learning extracts input features and labels output. This step requires domain knowledge to produce high-quality, and well-prepared data [11].

In wireless sensor networks, we favor local features (within one-hop) that can be collected without expensive communications. This is because sensor networks are severely resource-constrained and it is desirable and necessary to impose as little overhead as possible. However, if a feature is already available with the existing application, such as node depth in MintRoute, we also consider it. There is no extra overhead imposed to gather this feature and it carries extra useful information.

| Link quality learning | | |
|---|---|---|
| `RSSI` | received signal strength indication | local |
| `sendBuf` | send buffer size | local |
| `fwdBuf` | forward buffer size | local |
| `depth` | node depth from the base station | non-local |
| `CLA` | channel load assessment | local |
| `pSend` | forward probability | local |
| `pRecv` | backward probability | local |
| **Hop-by-hop caching learning** | | |
| `RSSI` | received signal strength indication | local |
| `pSend` | forward probability | local |
| `pRecv` | backward probability | local |
| `LQI` | link quality indication | local |

TABLE I
FEATURE VECTOR ILLUSTRATION.

**Feature selection.** This is the process of choosing a subset of the feature space that best represents the problem at hand while introducing a minimal amount of noise.

As pointed out in previous studies, link delivery probability (or link quality) is determined by many factors, including wireless channel conditions, such as internode separation, fast fading and slow fading, the traffic pattern in the network and local traffic load at each node, etc. However, the extent to which these factors impact link quality is continuously varying, which makes it impossible for any single metric to be always a good indicator of link quality. For example, Aguayo et al. [12] find that SNR (Signal/Noise Ratio), though affecting link delivery probability, cannot be expected to be a predictive indicator of link quality. Thus, we choose a set of metrics that are correlated to link delivery probability to be included in the feature vector and use machine learning tools to train and identify the most predictive indicator, which could be a combination of them. Some of the features are related to channel conditions, some of them related to network congestion, and some of them to both. Table I lists the features we used for link quality learning and hop-by-hop caching learning, respectively. They are all numerical values.

RSSI is the received signal strength indication readily available in many radios. It contains the average RSSI level during receiving of a packet in CC2420 with its value appended to each frame. RSSI is averaged over 8 symbol periods ($128\mu s$) and is continuously updated for new symbols received. Using the RSSI value directly to calculate the LQI value has several disadvantages [13]. In CC2420, LQI is not estimated based on RSSI, but rather a correlation value that indicates "chip error rate".[2] Therefore, RSSI may contain information that is not available in LQI and we include it as input feature here.

Channel load assessment is a metric used in CODA [14] to detect local network congestion. It uses a sampling scheme to monitoring local channel at appropriate time to minimize the energy cost while performing accurate estimates of congestion conditions.

Queue management is widely used in wired networks for congestion detection. In wireless networks, it is also closely related to local channel conditions. We use both forward buffer size and send buffer size as indications of congestion here. However, as pointed out in [14], without link-level acknowledgments, buffer occupancy or queue length cannot be used as an indication of congestion. In our experiment, link-level acknowledgment is enabled for the CC2420 radio.

Because network topology may strongly influence the traffic load in a data collection application, it could also have an impact on link delivery capability. Network topology can be characterized as node depth in a network or the number of children a node has in the data collection tree. We use node depth, which is defined as the number of hops to the sink in the collection tree. Due to funneling effects, node depth is strongly correlated to link quality.

Lastly, $pSend$ and $pRecv$ are originally used to derive the forward and backward delivery probability. Therefore, they capture important link quality information. On one hand, if their values are valid, they contain history information of link delivery. On the other hand, if their values are invalid, something unexpected has happened in the network, such as a congestion collapse, which could also be used to infer link quality. Therefore, we include them as input features. We will show later in this section that these two metrics are crucial in improving the classification accuracy.

Note that some features vary at a large time-scale while others vary more frequently. Since features are only meaningful when measured at an appropriate time-scale, we retain the originally-proposed time scales, such as RSSI on a per-packet time scale and CLA on a per-sampling scale. Such a combination of input features may not convey a meaningful metric in reality. However, we believe that they more aptly capture link quality than any other single metric alone.

**Output labeling.** Output labeling is the process of labeling sample outputs using domain knowledge. Supervised learning algorithms need to use labels to determine what class the input features are assigned.

There are many ways to label link quality based on LQI. We study two approaches in this paper. The first one uses a *binary* model that only predicts a link as "good" or "bad". The second one uses a *multi-class* model and can predict multiple classes of link quality. These link quality categories can distinguish link quality in a finer granularity than using the binary model. To one extreme, the multi-class model can predict the actual LQI value numerically, which then becomes a regression problem.

## *B. Step 2: Sample Collection*

To perform offline training, we collect samples from all nodes to a backend server. To avoid interference of sample collection traffic to regular application traffic, we send sample data to the programming board attached to each sensor node, as configured in MoteLab. If there is no programming board attached, or if the sensor nodes are deployed in an environment where such configuration is impossible, we can inject extra sensor nodes, or virtual sinks [15] that are used exclusively for siphoning the sample collecting traffic.

Since link quality is strongly correlated with data traffic in the network, we collect samples from a variety of offered

[2]A combination of RSSI and correlation values may be used to generate LQI, as suggested in [13].

|  | Predicted class |  |  |  |
|---|---|---|---|---|
|  | a | b | c | Total |
| a | 1456 | 257 | 26 | 1739 |
| b | 403 | 1369 | 124 | 1896 |
| c | 86 | 154 | 1586 | 1826 |
| Total | 1945 | 1780 | 1736 | 5461 |

TABLE II

CONFUSION MATRIX OF A THREE-CLASS CLASSIFIER USING JRIP.

|  | **JRip** |  | **J4.8** |  |
|---|---|---|---|---|
| Class | TP rate | FP rate | TP rate | FP rate |
| a | 0.837 | 0.131 | 0.841 | 0.133 |
| b | 0.722 | **0.115** | 0.712 | **0.103** |
| c | 0.869 | **0.041** | 0.885 | **0.046** |

TABLE III

DETAILED ACCURACY BREAKDOWN FOR ALL CLASSES.

load, ranging from 0.25 pps to 4 pps, in order not to lose traffic-related information. However, the number of samples collected from a non-congested network is far more than that collected from a congested network, using the same sample collection period. Hence, we choose to prolong the sample collection periods for high-load traffic so that we can have enough samples from a range of loads.

## C. Step 3: Offline Training

Our learning and validation experiment is performed on Weka [11], a workbench with implementations of a variety of standard machine learning algorithms. We use the J4.8 algorithm provided with Weka for decision tree learning and JRip algorithm for rule learning. J4.8 implements an improved version of the widely-used C4.5 algorithm and JRip [16] implements Repeated Incremental Pruning to Produce Error Reduction (RIPPER), a propositional rule learner.

As with most data-intensive machine learning algorithms, it is important to avoid having the classifier memorize, or overfit, the training data. We use cross validation and tree pruning in Weka to reduce such effects. Cross validation is a standard method to estimate classification accuracy over unseen data. We use 10-fold cross validation in our experiments. The available data is divided into ten equal-sized blocks. Nine of the blocks are randomly chosen and used for training a classifier, with the remaining block used for validation. This process is repeated 10 times to give a reliable measure of classification accuracy, which is 82% using J4.8 and 80% using JRip for our evaluation on MoteLab.

Table II shows the *confusion matrix* for a three-class prediction, in which class $a$ contains links with the best quality, class $c$ the worst and class $b$ in between. A confusion matrix is often used to display the cost and accuracy of a multi-class prediction. Each element (x,y) in the matrix shows the number of samples for which the actual class is x and the predicted class is y. The numbers down the main diagonal are those that are predicted correctly. The accuracy of our classifier is then $(1456 + 1369 + 1586)/5461 = 80.8\%$.

Table III shows the TP rate and FP rate of a three-class classifier for both JRip and J4.8, using the same dataset with 10-fold cross validation. For both algorithms, the FP rate of class $c$ is lower than 5%, which means that the probability of classifying a bad link as either a good or median one is low. In metric-based routing, the cost of such mis-classifications is high and both JRip and J4.8 work well in this aspect.

|  | **JRip** |  | **J4.8** |  |
|---|---|---|---|---|
|  | Binary | Multiple (3) | Binary | Multiple (3) |
| Accuracy | 82.6% | 80.8% | 85.2% | 81.1% |
| Overhead | 7 rules | 16 rules | 77 nodes | 135 nodes |
| FP rate | 5.9% | 4.1% | 11.3% | 4.6% |

TABLE IV

COMPARISON BETWEEN A BINARY AND A MULTI-CLASS CLASSIFIER.

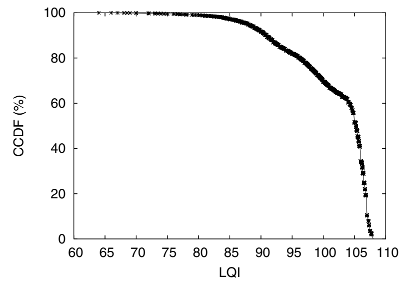


Fig. 2. Complementary CDF of LQI. The training set is collected from MoteLab. It shows that there is no clear threshold differentiating "good" links from "bad" ones.

## D. Discussions

**Binary or multi-class classifier.** One key difference between a binary classifier and a multi-class classifier is the flexibility in interpreting the labels. A link with median quality will either be classified as "good" or "bad" with a binary classifier. If the link is good but classified as bad, it will not be utilized. If we skew the threshold to treat more samples as good, the probability of not distinguishing between really good and fairly good links will increase. Using a multi-class classifier, however, produces additional information beyond a binary classifier. The extreme is a numerical classifier that predicts the exact LQI value. However, usually the accuracy drops with increased number of categories. The tradeoff is illustrated in Table IV, which compares the accuracy, memory footprint and FP rate between a binary classifier and a three-class classifier.

The accuracies of three-class classifiers using both JRip and J4.8 are lower than the accuracies of their corresponding binary classifiers by at most 3%. For J4.8, the size of the decision tree learner is relatively large. For JRip, however, the size increase is small since 16 rules take only a few hundred bytes to represent.

Another advantage of using multi-class classifiers is the small FP rate compared to a binary classifier. This can be explained by looking at the distribution of LQIs. Figure 2 is the Complementary CDF (CCDF) of the LQIs used in our dataset. It shows that there is no clear threshold that can divide the links into "good" ones and "bad" ones. Simply drawing a line to divide the links into two categories may result in classifying some links with median quality as "bad" ones and vice versa. The overall performance will suffer if such error rates are high. We will show in Section V how multi-class

| M1 | | M2 | |
|---|---|---|---|
| Rank | Feature | Rank | Feature |
| 0.70812 | psend | 0.3251 | RSSI |
| 0.58138 | RSSI | 0.1577 | fwd_buf |
| 0.34003 | precv | 0.1384 | psend |
| 0.03586 | depth | 0.0771 | precv |
| 0.00406 | fwd_buf | 0.0628 | depth |
| 0 | CLA | 0 | send_buf |
| 0 | send_buf | 0 | CLA |

TABLE V
RANKED ATTRIBUTES.

| | **7-feature** | **5-feature** | **1-feature (RSSI)** | **1-feature (pSend)** |
|---|---|---|---|---|
| Accuracy | 80.8% | 80.8% | 70.5% | 69.3% |
| Overhead | 16 rules | 17 rules | 4 rules | 20 rules |
| Bad FP rate | 4.0% | 4.1% | 3.9% | 4.1% |

TABLE VI
IMPACT OF FEATURE SELECTION. THE 5-FEATURE SET IS SELECTED USING THE UNION OF FEATURES IN M1 AND M2.

classification can improve routing performance.

**Feature selection.** Because irrelevant features will degrade the performance of decision tree learners and rule learners [11], it is beneficial to perform an attribute selection to eliminate all but the most relevant features. We already selected a set of features based on our understanding of the problem domain and the physical meaning of each attribute. Here, we use some well-established methods to further sieve those features to further improve prediction accuracy and reduce the overhead of feature collecting.

We use two attribute selectors provided by Weka: InfoGainAttributeEval (M1) and GainRatioAttributeEval (M2). We use the union of their output features as our final feature vector, as shown in Table V.

M1 evaluates the worth of an attribute by measuring the information gain with respect to the class, while M2 measures the gain ratio. M2 takes into account the information each attribute contains, which is neglected in M1. Equations 1 and 2 are their mathematical definitions:

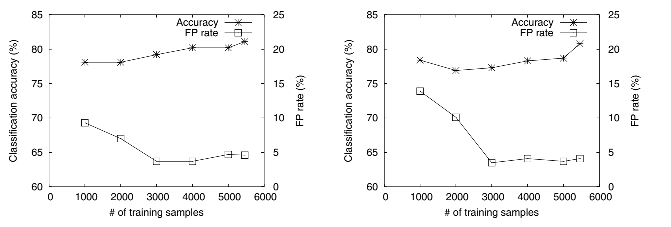
$$InfoGain(C, Attr) = I(C) - I(C|Attr) \quad (1)$$

$$GainRatio(C, Attr) = \frac{I(C) - I(C|Attr)}{I(Attr)} \quad (2)$$

with $I(C|Attr)$ the binary entropy of class $C$ given attribute $Attr$. Entropy is widely used in machine learning to represent the amount of information an attribute contains with respect to the class of interest.

The impact of feature selection to learning accuracy, memory footprint and FP rate of class $c$ (bad) is demonstrated in Table VI. In particular, we compare the accuracy using all 7 features to the accuracy of using only one feature. Clearly, using more features results in a higher accuracy than using only one. This supports our motivation to look at more features. The impact of feature selection to routing performance is discussed in Section V.

**Impact of training corpus size.** Figure 3 shows how training corpus size affects classification accuracy and FP rate. Empirically, 5000 samples are sufficient.



(a) J4.8 (b) JRip

Fig. 3. Accuracy (left-axis) and FP rate (right-axis) as a function of training corpus size.

**Hardware dependency.** Although our supervised classification process requires a manual method to label link quality, it does not depend on any particular metric, such as the LQI value available only on 802.15.4 radios. Other metrics, if indicative of link quality, can also be used for labeling.

## IV. CASE STUDIES

In this section, we present case studies that illustrate how supervised learning technique can be leveraged to enable situation-aware routing in wireless sensor networks.

### A. *Usage I: Collection Routing in Congested Networks*

MintRoute is a collection routing protocol that uses ETX to construct a routing tree to the sink. It fails to find parents in congested networks due to the malfunction of snooping-based link quality estimation. We propose MetricMap, an alternative to MintRoute that establishes link quality estimations using offline trained classifiers to address this problem.

MetricMap consists of two components. The first component controls the update of all features which is triggered either by packet arrivals or timer events. The second component controls link classification, with input from features collected by the other component and output in numerical or categorical values indicating link quality. The output of the classifier is used whenever the ETX-based method fails.

### B. *Usage II: Situation-Aware Reliable Transport (SHARP)*

Two major causes of packet drops in wireless sensor networks are lossy links and a constrained storage hierarchy. We have discussed link transmission characteristics in the previous case study. As to storage hierarchy, sensor nodes are severely constrained in the amount of memory available to applications and this problem is compounded by the lack of dynamic memory allocation in TinyOS. Advances in hardware design may mitigate this problem [17]. However, the storage hierarchy will still be constrained for many applications and passive caching/retransmission mechanisms used in reliable transport can lead to resource waste and contention. We present SHARP, a situation-aware reliable transport atop PSFQ that considers the factors correlated with packet retransmissions to improve resource efficiency while maintaining desired reliability.

SHARP also works in two phases. In the training phase, all nodes run PSFQ and a data logger is used to collect the feature vector and the fetch event. The data logger is a PC connected to the MicaZ nodes via a programming board. The mote is either triggered by a timer or by a fetch event. We use a timer

```
// update feature vector on demand or periodically
void updateRSSI () {
  foreach packet successfully received from neighbor i
    keep the RSSI value history for i
}
void updateBuf (int type) {
  during each update interval
    update the buf size for type (fwdBuf or SendBuf)
}
void updateCLA () {
  during each update interval
    check the clear channel assessment and update CLA
}
void updateProbSend () {
  // this feature is updated the same as in MintRoute
}
void updateProbRecv () {
  // this feature is updated the same as in MintRoute
}
int classify (struct featureVec fv) {
  // perform classification based on input features
  // the output represents the class label
}
// update link quality based on classification results
// recvEst is the in-bound link quality estimation
// link quality is between 0 (low) and 255 (high)
void updateEst(fv) {
  if (classify(fv.rssi, fv.sendBuf, fv.fwdBuf, fv.depth,
               fv.CLA, fv.pSend, fv.pRecv) == "good") {
    recvEst = 1 * 255
  } else {
    recvEst = 0
  }
}
```

Fig. 4. Pseudo-code of MetricMap.

to trigger creations of training samples when no fetch event occurs. This is to ensure that we have enough samples from both categories: *fetch* or *no fetch*. After collecting training samples, we conduct an offline learning that outputs a classifier predicting *fetch* or *no fetch* given the features observed. In the online phase, we embed this classifier into PSFQ. Only when the classifier predicts a fetch shall we cache those packets this node pumps out recently.

Since multiple packets could be fetched when a packet loss is detected, we need to ensure that packets arrived recently are cached locally to serve this fetch. We look at the cache size as a sliding window and define it as *history window*. Intuitively, the larger the history window, the lower the fetch miss rate and the higher the memory footprint. In our current design, we use a constant as the history window size.

### C. Implementation

The pseudo-code is listed in Figure 4. All functions starting with `update` are used for online feature collection. As we discussed in previous sections, we need to collect a set of features with different features at different time scales. For example, `update{Buf,CLA,ProbSend,ProbRecv}()` are all at a per-sampling scale and `updateRSSI()` is at a per-packet scale. Similar to MintRoute, MetricMap periodically updates its estimations of link quality in `updateEst()`. It returns a value between 0 and 255 that is used to calculate the ETX from a node to the sink. In the core of `updateEst()` is a function call to `classify()`, which implements the classification component and returns an estimate of link quality in categories using our offlined learned classifier. This is significantly different from MintRoute in that no packet snooping is required to conduct link quality estimation.

Due to space limitation, we omit a description of the SHARP implementation. However, the classifiers used in both cases are very similar and the only difference is in the type of features used.

## V. Testbed Evaluation

To illustrate the application of supervised machine learning in realistic sensor network application settings, we have implemented a collection routing protocol and a reliable transport in TinyOS and deployed them on testbeds of real sensor nodes.

In what follows, we present evaluation results of our MetricMap prototype deployed over a real-world sensor network testbed and our SHARP prototype deployed over a multihop sensor network in our lab. They both use TinyOS and run on MicaZ motes.

### A. Evaluation Methodology

In our evaluation, we consider the following performance metrics:

*Data delivery rate:* The fraction of data packets that are successfully delivered to the destination.

*Data latency:* The time it takes to send a packet out till the packet is received at the sink.

*Fairness index:* This metric [18] is used to measure the variability of performance across all source nodes. This is an important metric for applications that require same delivery performance from all sources. For any given set of delivery rates $(p_1, \ldots, p_n)$, the fairness index definition adapted for our problem is given by:

$$f(p_1, \ldots, p_n) = \frac{(\Sigma_{i=1}^n p_i)^2}{n\Sigma_{i=1}^n p_i^2}$$

with $p_i$ denoting the average packet delivery rate of the $i$th sensor and $n$ the total number of source nodes in the network. The fairness index always lies between 0 and 1. If all nodes have the same packet delivery rate, the fairness index is 1.

In each experiment, we also measure the overhead required to achieve these performance metrics. In particular, we are interested in **memory footprint**. The testbed is comprised of MicaZ motes which have ATMEL 7.37 MHz ATMega128L, low-power, 8-bit micro-controller with 128 KB of program memory, 512 KB measurement serial flash data memory, and 4 KB EEPROM. It uses a Chipcon CC2420,a single-chip IEEE 802.15.4 compliant Radio Frequency (RF) transceiver operating at 2.4 GHz and capable of transmitting at 250 kbps. The packet size used in our experiments is 29 bytes, the default value in TinyOS. These motes are connected to an Ethernet for logging and mote-programming.

### B. MetricMap Results

We evaluate the performance of MetricMap on the MoteLab testbed, consisting of 30 motes across multiple offices in the Harvard Computer Science Building.

Our experiment consists of two phases: the offline learning phase, which takes multiple hours for collecting training samples and processing the learning task using Weka; and the online optimization phase that uses the inference rule learned in the training phase to drive situation-aware routing. Each run lasts 15 minutes.

When we evaluate the performance of MintRoute and MetricMap on MoteLab, the results are different for runs at different times. This is because of uncontrollable factors in the testbed, especially the variability of link qualities. Therefore, we take the following approach to reduce the impact of uncontrollable factors in the environment. We run MintRoute followed by MetricMap or vice versa for a continuous 15 minutes. We run such pairs of experiment 5 times and each experiment is independent with respect to each other. Such a design allows us to minimize influences from factors other than the algorithm itself. Also, our experiments are performed both in daytime and nighttime when the human activity interference decreases. For each offered load, the minimum, median and maximum values are shown.

**Performance and Overhead.** Figure 5(a) compares the data delivery rate between MetricMap and MintRoute. Our approach, MetricMap, consistently outperforms MintRoute. The higher the traffic load, the better MetricMap performs compared to MintRoute. MintRoute can rarely form a data collection tree under high traffic rates. In contrast, our approach can form a tree because it does not rely on data traffic for link quality assessment.

Figure 5(b) shows the packet latency comparison. Packets delivered by MetricMap have a comparable average latency to those delivered by MintRoute. Data latency includes local processing time at the source node and all intermediate nodes along a multihop route, network transmission time over all links and reception processing time at destination. Our classifier will be used regularly for updating the data collection tree. This may introduce some delay in the local processing time and transmission time if the calculation is on the critical path of data transmission. Our results show that the extra processing time in classification online does not impose a high overhead and delay on packet transmission.

Figure 5(c) compares the fairness index of packet delivery. It demonstrates that our approach is much better able to maintain fairness across different offered loads. It does not treat certain nodes better than others. This is reasonable since all nodes use similar rule-sets learned offline and there is no bias towards any particular link. On the other hand, since MintRoute relies on data traffic to infer link quality, the link selected may be skewed depending on the traffic pattern and their location to the sink. If any part of the network en route to the sink is overloaded, the MintRoute data collection process will be interrupted. MintRoute uses broadcast in this case to try to resume the communication, but this actually exacerbate the problem by adding more useless traffic into the network. Our classifier can mitigate the problems by discerning meaningful link information without imposing any additional traffic. Once the routing tree is re-formed, the data collection process can be resumed very quickly. So, using MetricMap, more nodes can deliver their data to the sink, which results in a higher fairness index. In contrast, MintRoute has a few nodes that deliver a lot of packets and the rest that have a very low success rate.

In summary, MetricMap addresses the high data rate challenge with a different perspective, compared to congestion control mechanisms [14], [15], [19]. Our approach is orthogonal to theirs and combining them will potentially achieve further performance improvement.

| Component | ROM (Flash) | RAM |
|---|---|---|
| Surge+MintRoute | 16570 | 1971 |
| Surge+MetricMap | 18468 | 2110 |

TABLE VII
CODE AND MEMORY USAGE COMPARISONS ON MICAZ. RAM IS MEMORY USAGE IN BYTES AND ROM IS PROGRAM SIZE IN BYTES.

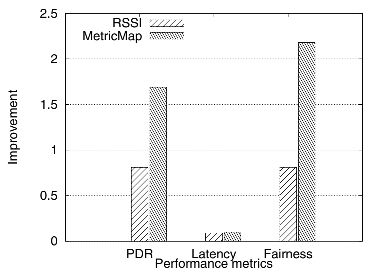
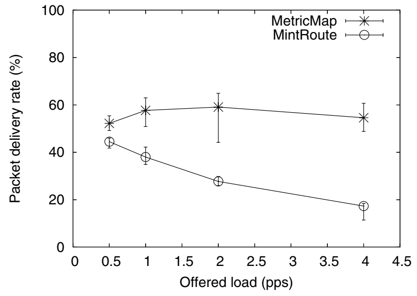


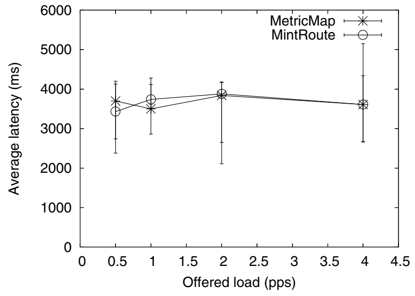
Fig. 6. Performance improvement comparison with heuristics-based approach.

Since MetricMap needs to keep local metrics that are used as input to the classifier, it requires some extra memory usage. We use the memory footprint of MetricMap as a measure of overhead, as shown in Table VII. Table VII shows the actual memory footprint of MintRoute and MetricMap. The increase in program size is 11.5%, which is used mostly for implementing the classifier. The increase in static memory size is 7.1%, which is mostly data structures used for collecting and converting low-level metrics to input of the classifier. This is a small increase from the original code and memory footprint.

Our results so far have shown that MetricMap produces consistently higher performance than MintRoute when the traffic rate is high. To understand if such benefits come from a better selection of good quality links, we further compare MetricMap with another data collection protocol *RSSI*, which uses the RSSI values of received packets as the only indication of a link's quality. If the recently received packets have higher RSSI values compared to other links, the protocol will assign a higher quality value to this link than the others. Other than that, RSSI is the same as MetricMap. Thus, RSSI does not account for any factors other than packet RSSI values and makes its estimation using heuristics.

Figure 6 shows the average improvement of RSSI and MetricMap over 5 independent testbed runs, using the performance of MintRoute as the base line. For example, the improvement of protocol $RSSI$ in terms of packet delivery rate is calculated as $(p_{\mathrm{RSSI}} - p_{\mathrm{MintRoute}})/(p_{\mathrm{MintRoute}})$. The figure shows that MetricMap has a higher performance in terms of packet delivery rate and fairness index, compared to RSSI. Since MetricMap uses more features to make link quality estimation, it potentially will find better links that have the capability to deliver more traffic. There is a minor increase in data latency for both protocols. This is because both RSSI and MetricMap deliver more packets than MintRoute and these packets usually have longer number of hops to traverse.


(a) Average success rate.


(b) Average packet latency.

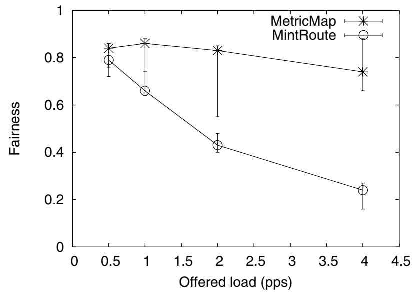

(c) Fairness index.

Fig. 5. Performance versus per-sensor load using a periodic workload.

## C. SHARP Results

We ported the initial PSFQ release to the MicaZ platform and tested it using TinyOS 1.1.13. We also linked the parent-selection component of MintRoute to PSFQ to get link quality assessment. The transmission radii are configured to form an unreliable 2-hop multihop network in an indoor office environment. We send a 20KB file using the two-hop network. Due to the long-term temporal variability of link quality [20], transferring a short file may result in two outcomes: (1) the link quality is constantly low and packet fetches are very frequent, or (2) the link quality is constantly good and packet fetches are very rare. In the first case, SHARP performs as well as PSFQ as it caches almost every packet. In the second case, SHARP performs as well as PSFQ in terms of reliability but with a much lower memory usage. Therefore, to fairly compare between PSFQ and SHARP, we need a situation that demonstrates such variability. On average, SHARP should outperform PSFQ since it saves memory for periods that have good connectivity. Furthermore, resource efficiency becomes more important when distributing large files. The data file is divided into blocks of 16 bytes to be pumped in 29-byte packets. Therefore, each cache line saved in flash memory is 16-byte long.

To understand the performance of SHARP, we use the following performance metrics: (1) **Miss rate of fetch**, defined as the ratio of the number of misses to the total number of (re)fetches, and (2) **Storage efficiency**, defined as:

$$\text{Storage efficiency} = \frac{M_{\text{PSFQ}} - M_{\text{SHARP}}}{M_{\text{PSFQ}}}$$

with $M_p$ the memory footprint of protocol $p$.

**Performance results.** We used a history window of 192 bytes, which corresponds to 12 cache lines. The miss rate of fetch is only 4.7% for this experiment. Thus, only 95.3% of the fetches can be repaired locally and only 4.7% of the fetches need to be saved by nodes at least two hops away. In applications that require large amount of space, our scheme will produce significant savings in storage. The saved space could either be used for storing data from other nodes to increase reliability, or for optimization purposes, such as in-network processing, compression, etc. In a high rate, many-to-one structure monitoring application, we conjecture even higher benefits due to the data funneling effect.

| Component | ROM (Flash) | RAM |
|---|---|---|
| PSFQ [21] | 22752 | 1163 |
| SHARP [3] | 26842 | 2724 + 16N |

TABLE VIII
CODE AND MEMORY USAGE COMPARISONS ON MICAZ. N IS THE ADDITIONAL NUMBER OF FEATURES TO COLLECT.

**Overhead.** Table VIII shows the code size and memory footprint of both PSFQ and SHARP. The code size of SHARP is 18% larger than PSFQ. This mainly comes from the feature collection part, which also includes the link quality assessment component of MintRoute. Since the total program size in MicaZ is 128K bytes, the amount of increase is negligible. The memory footprint of SHARP is 1561 bytes more than PSFQ, plus the memory required for any additional feature to collect. Each feature can be represented using any data structure. In SHARP, all features are 16-bit integers. The major increase comes from the parent-selection component of MintRoute. Since we only need the link quality assessment function, the memory footprint can be further optimized to drop the unnecessary components.

## VI. Related Work

Significant work has been done to achieve the ability to rapidly observe, decide and react to the dynamics in wireless sensor networks, where a wide range of network conditions exist. Most previous work either uses "rule of thumb" focusing on a single metric that may lose useful information or mislead the understanding of situations, or uses sophisticated heuristics that takes a lot of expertise and domain knowledge to derive. This section surveys the most related work in this aspect within sensor networks. We also briefly review the application of machine learning to problems in other domains.

**Link quality estimation.** Link quality awareness permeates many aspects of sensor network design and operation, ranging from the design of MAC protocols to the design of applications. As a result, link quality estimation has become a significant research focus. Many of the proposed metrics [20], [22]–[24] consider spatial or temporal variability of link quality. Unfortunately, they share one similar limitation: the

[3]The version of SHARP evaluated is not optimized for program size and memory footprint, but rather ease of feature collection and code readability.

performance of their metrics depends heavily on model accuracies, which need trial-and-error tuning and expert knowledge. Our approach, on the other hand, passively collects features and uses standard learning algorithms to discover the inner correlation. Furthermore, their observations on temporal and spatial variability of channel conditions can be used in our work to improve learning efficiency.

**Machine learning.** There has been significant prior work applying machine learning to different areas of research, including system-related problems, such as compiler optimizations [25], and reliability optimizations [26]–[28]. Machine learning has also been used for modeling data generated by sensor networks. Guestrin et al. [29] used kernel-based regression to accurately model sensor data and reduce the dimensionality of data representation. This approach significantly decreases the communication requirements in the network. More recently, Krause et al. [30] studied sensor placements using probabilistic models that account for both data quality and communication costs. Our approach, however, focuses on optimizations within the networking protocol stack.

**Routing optimization.** In terms of efficient routing design in the presence of unreliable radio links, [31] takes a joint-optimization approach that considers both the recovery of lost packets in the link layer as well as path selection in the routing layer. The metric they proposed considers many of the features we used in this work. However, our focus is on learning information that is otherwise unavailable with traditional approaches. Therefore, our method can be combined with theirs to further improve communication efficiency.

## VII. Conclusions and Future Work

This paper presents a supervised learning framework that helps to make informed routing and reliability decisions in sensor networks. We have applied this framework to both link quality estimation and to packet caching with good results. In addition to performance improvements, our approach also mitigates the complexity often needed to heuristically incorporate situation awareness into network systems.

Beyond this initial prototype, we envision future work to include the following. First, we wish to incorporate more dynamic and online training techniques that can better adjust to varying network conditions. Second, we wish to evaluate how our approaches work in very heterogeneous networking conditions, where distributed learning may be desirable. Third, we wish to apply our learning techniques to other network optimization problems.

Overall, this work offers an important first look at machine learning techniques for the particular network problems we have evaluated. In demonstrating machine learning's considerable performance advantages, this paper has made a first step towards clean implementations of highly-effective, situation-aware learners for a variety of challenged networks.

## References

[1] D. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," in *Proc. ACM MobiCom*, 2003.

[2] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in *Proc. ACM SenSys*, 2003.

[3] H. Zhang, A. Arora, and P. Sinha, "Learn on the fly: Data-driven link estimation and routing in sensor network backbones," in *Proc. IEEE INFOCOM*, 2006.

[4] J. Paek, K. Chintalapudi, R. Govindan, J. Caffrey, and S. Masri, "A wireless sensor network for structural health monitoring: Performance and experience," in *Proc. IEEE Workshop on Embedded Networked Sensors (EmNetS)*, 2005.

[5] L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis, "Design and deployment of industrial sensor networks: Experiences from the north sea and a semiconductor plant," in *Proc. ACM SenSys*, 2005.

[6] V. Shnayder, B. rong Chen, K. Lorincz, T. R. F. Fulford-Jone, and M. Welsh, "Sensor networks for medical care," Harvard University, Tech. Rep. TR-08-05, 2005.

[7] *IEEE Standard 802, part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPANs)*, 2003.

[8] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," in *Proc. IPSN/SPOTS*, 2005.

[9] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy, "Pump-Slowly, Fetch-Quickly (PSFQ): A reliable transport protocol for sensor networks," *IEEE Journal on Selected Areas in Communications*, 2005.

[10] T. Mitchell, *Machine Learning*. McGraw Hill, 1997.

[11] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, 2005.

[12] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-level measurements from an 802.11b mesh network," in *Proc. ACM SIGCOMM*, 2004.

[13] "Chipcon CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver," http://www.chipcon.com/.

[14] C.-Y. Wan, S. B. Eisenman, and A. T. Campbell, "CODA: congestion detection and avoidance in sensor networks," in *Proc. ACM SenSys*, 2003.

[15] C.-Y. Wan, S. B. Eisenman, A. T. Campbell, and J. Crowcroft, "Siphon: Overload traffic management using multi-radio virtual sinks," in *Proc. ACM SenSys*, 2005.

[16] W. W. Cohen, "Fast effective rule induction," in *Proc. the International Conference on Machine Learning (ICML)*, 1995.

[17] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy, "Ultra-low power storage for sensor networks," in *Proc. IPSN-SPOTS*, 2006.

[18] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. John Wiley & Sons, Inc., 1991.

[19] B. Hull, K. Jamieson, and H. Balakrishnan, "Mitigating congestion in wireless sensor networks," in *Proc. ACM SenSys*, 2004.

[20] A. Cerpa, J. Wong, M. Potkonjak, and D. Estrin, "Temporal properties of low-power wireless links: Modeling and implications on multi-hop routing," in *Proc. ACM MobiHoc*, 2005.

[21] PSFQ 0.10 Release, http://www.comet.columbia.edu/armstrong/release/release.html.

[22] C. E. Koksal and H. Balakrishnan, "Quality-aware routing in time-varying wireless networks," *to appear in IEEE Journal on Selected Areas of Communication Special Issue on Multi-hop Wireless Mesh Networks*, 2006.

[23] A. Cerpa, J. L. Wong, L. Kuang, M. Potkonjak, and D. Estrin, "Statistical model of lossy links in wireless sensor networks," in *Proc. IPSN*, 2005.

[24] Y. Xu and W.-C. Lee, "Exploring spatial correlation for link quality estimation in wireless sensor networks," in *Proc. IEEE PerCom*, 2006.

[25] B. Calder, D. Grunwald, D. Lindsay, M. Jones, J. Martin, M. Mozer, and B. Zorn, "Evidence-based static branch prediction using machine learning," *ACM Transactions on Programming Languages and Systems*, January 1997.

[26] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. Chase, "Correlating instrumentation data to system states: A building block for automated diagnosis and control," in *Proc. OSDI*, 2004.

[27] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *Proc. IEEE ICAC*, 2004.

[28] B. Liblit, A. Aiken, A. X. Zheng, and M. I. Jordan, "Bug isolation via remote program sampling," in *Proc. ACM PLDI*, 2003.

[29] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden, "Distributed regression: an efficient framework for modeling sensor network data," in *Proc. IPSN*, 2004.

[30] A. Krause, C. Guestrin, A. Gupta, and J. Kleinberg, "Near-optimal sensor placements: maximizing information while minimizing communication cost," in *Proc. IPSN*, 2006.

[31] Q. Cao, T. He, L. Fang, T. Abdelzaher, J. Stankovic, and S. Son, "Efficiency centric communication model for wireless sensor networks," in *Proc. IEEE INFOCOM*, 2006.