A Supervised Learning Approach for Routing Optimizations in Wireless Sensor Networks

Yong Wang, Margaret Martonosi, and Li-Shiuan Peh
Princeton University
{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. In this paper, we explore using supervised learning techniques to address such challenges in wireless sensor networks. Machine learning has been very effective in discovering relations between attributes and extracting knowledge and patterns using a large corpus of samples.

As a case study, we use link quality prediction to demonstrate the effectiveness of our approach. For this purpose, we present MetricMap, a link-quality aware collection protocol atop MintRoute that derives link quality information using knowledge acquired from a training phase. Our approach allows MetricMap to maintain efficient routing in situations where traditional approaches fail. Evaluation on a 30-node sensor network testbed shows that MetricMap can achieve up to 300% improvement on data delivery rate in a high data-rate application, with no negative impact on other performance metrics, such as data latency. Our approach is based on real-world measurement and provides a new perspective to routing optimizations in wireless sensor networks.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Routing protocols*; I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Design, Experimentation, Measurement, Performance

Keywords

Sensor networks, link quality, supervised learning, classification

1. INTRODUCTION

Many critical applications in wireless sensor networks rely very 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 network adaptations are twofold. 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 communication. 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. 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 two phases: an offline training phase followed by an online classification. Offloading the training task from the sensor node reduces the processing, communication, and energy requirements of the node. The resulting classifiers to be used online are both strikingly lightweight and strikingly effective. For the case studies we have examined, our supervised learning techniques result in prediction accuracies of 80% or more, with false positive rates between 4.1% and 11.3%, and with essentially no compute overhead during their online phase.

We evaluate the effectiveness of our approach using link quality prediction as a case study. For this purpose, we present MetricMap, a data collection protocol atop MintRoute that predicts link quality using knowledge gathered at the training phase when the network is highly congested. Evaluation of a prototype implementation in TinyOS on a real-world sensor network shows that MetricMap can improve over existing approaches by up to a factor of 3 for a high data-rate application. The compactness of our classifier makes it suitable for resource-constrained situations.

The primary contributions of this paper are summarized as follows. First, we develop a framework that uses supervised learning to automatically extract useful information within sensor networks. Because the method is automatic, our technique can be used for other situations having different hardware and other run-time factors with minor modifications. This is advantageous over heuristic methods whose effectiveness may depend on the context where they are developed and evaluated.

Second, we cast the link quality estimation problem as a classification problem, which permits the use of standard, yet effective algorithms. Decision tree learners and rule learners represent such algorithms. We believe a large range of applications can benefit from this approach.

Third, we show that tree-based routing topologies in data collection applications may suffer from information loss, such as neighbor link quality, in an overloaded network. We use supervised learning to establish data collection trees in such adverse conditions. Our approach is capable of maintaining efficient routing by locating high quality links.

The rest of this paper is organized as follows. Section 2 introduces the background knowledge. Section 3 describes the details of our learning framework. Section 4 and 5 present our case study and results of a prototype implementation on a real-world sensor network testbed. Related work is discussed in Section 6 and the last section summarizes the main results and outlines future work.

2. BACKGROUND

2.1 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.

2.1.1 Software-based Estimation

A few software-based link metrics have been proposed in the past. Route metrics are built atop these link metrics to capture the end-to-end capability of forwardness. For example, ETX [9], also proposed in MintRoute [26] is one such route metric. It 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. 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 [20, 15], a snooping-based method works poorly, as we will quantify shortly. For example, consider the high data-rate structure monitoring application discussed in [20]. Due to structural vibration damping effects, a very high data sampling rate is required, which is estimated to be at least 200Hz. Therefore, the data rate can be 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 data rate is still very challenging for current systems to cope with.

To demonstrate the impact of high traffic loads on ETX's link quality estimator, we evaluate the performance of MintRoute by running the Surge application[1] on MistLab [17], an indoor sensor network testbed of 60 Mica2 nodes. Surge is a data collection application in which each node generates data traffic at a constant rate and sends to the sink via multi-hop routing. We use MintRoute to build the multi-hop data collection tree that chooses a parent based on additive link/path quality estimation. Figure 1 shows that packet delivery rate degrades once the offered load is 2 packets/second (pps) or higher.

Figure 1(a) shows the network-wide fraction of orphan nodes, defined as nodes that have no parent information in the collection tree, with traffic loads of 2pps and 4pps. We only consider orphans caused by lack of information, instead of those caused by network disconnections. The percentage of orphan nodes increases quickly with increases of offered load. For 4pps offered load, 90% of the nodes do not have a parent 50% of the time. This dramatic increase in percentage of orphan nodes is a direct cause of data packet loss in the network, shown in Figure 1(c). Given a percentage of packets p received from a given node at the sink, the Cumulative Distribution Function (CDF) plots the fraction of sensors that deliver at most p percent of their data to the sink. For the 4pps case, about 60% of all nodes have less than 10% data delivered. Figure 1(b) plots the distribution of orphan nodes as a function of time. The x-axis is the experiment timeline in units of seconds. The y-axis is the node ID. Each square dot at (x, y) indicates that at time instant x, node y has no parent. In a network with a partitioned collection tree, many packets are transmitted from the edge toward the sink, only to be dropped before reaching the sink.

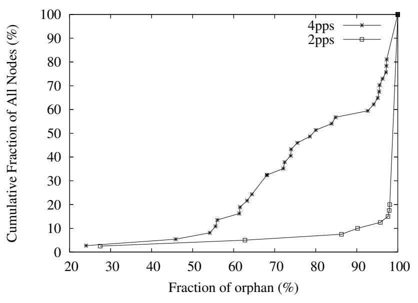
An examination of the etxs of all nodes shows that a large proportion of links have quality values indicating barely any transmissions can be carried through. This is directly related to how snooping-based estimation methods behave in an overloaded network. As a result, routing is interrupted due to a lack of link quality information. However, since not all links are overloaded, routing can be resumed once an accurate estimation of link quality is in place. We wish to develop link quality estimators that are more resilient in high-traffic settings. Machine learning offers us an efficiency way to discover them.

2.1.2 Hardware-based Estimation

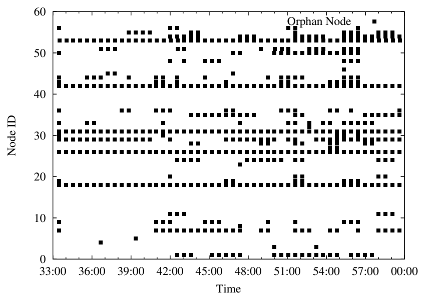
The link quality indication (LQI) metric is a characterization of the strength and/or quality of a received packet, introduced in 802.15.4 standard [1] and provided by CC2420, the radio used in many mote platforms, including MicaZ and Telos. LQI measures the incoming modulation of each successfully received packets and the result is an integer ranging from 0x00 to 0xff. The minimum and maximum LQI values (0x00 and 0xff) are associated with the lowest and highest quality signals detectable by the receiver (between $-100dBm$ and $0dBm$). Link quality values in between are uniformly distributed between these two limits. A measurement study [21] on the Telos platform shows that the average LQI closely maps the 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

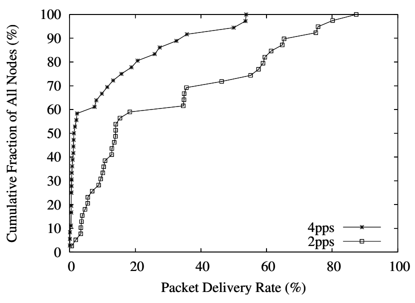
[1]The reference implementation is in the TinyOS CVS repository: `tinyos-1.x/apps/Surge/`.



(a) CDF of percentage of orphan nodes.



(b) Spatial distribution of orphan nodes as a function of time.



(c) CDF of packet delivery rate (PDR).

Figure 1: Experiment results in a MistLab testbed deployment (60 motes). The percentage of orphan is defined as the ratio of orphan period to the whole running time. We periodically probe the routing state of a node and estimate this fraction as the ratio of the number of times that the node is an orphan to the total number of probes. The first two figures shows the spatial and temporal distribution of orphan nodes for different offered load. The fraction of orphan nodes is very high when the offered load is above 2pps, which leads to a lack of routing information and a need for prediction.

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.

2.2 Supervised Learning Overview

The goal of supervised learning is to predict the value of an outcome measure based on a number of input measures [18]. 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 study and many times they achieve the highest accuracy among all algorithms studied.

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. Like decision tree learners, rule learners work on training samples with similar input/output pairs. 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 classifiers need to be used at runtime.

Learning overhead. Due to resource constraints in wireless sensor networks, however, we need to also consider learning efficiency and overhead, in addition to learning accuracy. These constraints include node processing time, energy budget, and memory footprint, etc.

Since in our proposed framework, training is conducted offline, usually on a resource-rich backend PC or server, we focus on the overhead of online classification and feature collection. 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 will result in hundreds of rules to be hand-coded into the protocol. Therefore, we instead prefer to use the output from rule learners in implementing the online classifier, if their accuracies are acceptable. Also, we prefer learners that produce human-readable output and both decision tree learners and rule learners are good for this purpose.

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 the FP rate represents the *cost* of learning. Usually we want a high TP rate (high benefits) and a low FP rate (low costs).

3. LEARNING STEP-BY-STEP

In this section, we introduce the steps of our learning framework. Figure 2 presents a high level overview of the steps involved, with the four key steps listed as follows:

1. First, we select the features to be used in training and classification;
2. Then, we instrument every node in the network to track these features and their corresponding labels which are periodically collected;

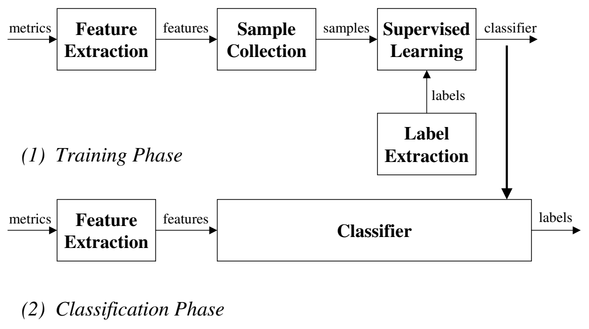


Figure 2: Overview of learning steps.

3. Next, we used the labeled data to perform training at the backend server;
4. Finally, we instrument MintRoute to use the classifier for differentiating between high quality and low quality links at runtime for real deployment. The algorithm is depicted in Section 4.

In what follows, we describe the first three steps listed above with specific reference to a collection routing application. Since the last step is closely tied to the application, we put the discussion of application instrumentation using classifiers to Section 4.

3.1 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 [25].

In wireless sensor networks, we favor local features (within one-hop) that can be collected without expensive communications. This is because sensor networks are very 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 from MintRoute, we also consider it. There is no extra overhead imposed to gather this feature and it carries extra useful information.

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 of 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. [2] 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 them are related to channel conditions, some of them related to network congestion, and some of them to both. Table 1 lists the features we collected for link quality learning. They are all numerical values.

RSSI is the received signal strength indication readily available in many radios. In CC2420, it contains the average RSSI level during receiving of a packet with its value appended to each frame. RSSI is continuously updated for new symbols received.

`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

Table 1: Feature vector illustration.

Channel load assessment is a metric used in CODA [23] to detect local network congestion. It uses a sampling scheme to monitor local channel conditions and minimize energy cost while performing accurate estimates of congestion conditions.

Queue management is widely used in wired networks for congestion detection. In wireless network, it is also closely related to local channel conditions. We use both forward buffer size (used for multi-hop forwarding) and send buffer size to track local congestion conditions. However, as pointed out in [23], 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, it could also impact link quality. Network topology can be characterized using metrics such as node depth or the number of children a node has in a collection tree. We use node depth here since it is strongly correlated to link quality due to data funneling effects.

Lastly, $pSend$ and $pRecv$ are metrics originally used to derive the average forward and backward probability. They capture important link quality information. On one hand, if their values are valid, they will contain history information of link delivery. On the other hand, if their values are invalid, they indicate that 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 classification accuracy.

Output labeling. Output labeling is the process of classifying sample outputs using domain knowledge. Supervised learning algorithms need to use labels to determine what category the input feature vector is 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 predicts a link either “good” or “bad”. The second one uses a *multi-class* model and can predict a set of classes of link quality. These categories can be used to distinguish link quality in a finer granularity than using the binary model. To one extreme, the multi-class approach can predict the actual LQI numerically, which becomes a regression problem.

3.2 Step 2: Sample Collection

To perform offline training, we collect samples from sensor 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 a configuration is impossible, we can inject extra sensor nodes or virtual sinks [24] that are used exclusively for siphoning off the sample collecting traffic.

Since link quality is strongly correlated with data traffic, we collect samples from a variety of offered load, ranging from 0.25 pps to 4 pps, in order not to lose traffic-related information. However,

	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 2: Detailed accuracy breakdown for all classes.

the number of samples collected from a non-congested network is far more than those collected from a congested network, with the same sample collection period. Hence, we use longer collection periods under high traffic loads in order to collect enough samples from a wide range of conditions.

3.3 Step 3: Offline Training

Our learning and validation experiment is performed on Weka [25], a workbench containing 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 C4.5 algorithm [22] and JRip [8] implements Repeated Incremental Pruning to Produce Error Reduction (RIPPER), a propositional rule learner. C4.5 is one of the most widely studied and used decision tree algorithms in the literature.

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 2 shows the TP rate and FP rate of a three-class classifier for both JRip and J4.8, using the same link quality estimation 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 the context of link-quality aware routing, the cost of such misclassification is high and both JRip and J4.8 work well in this aspect.

3.4 Discussion

Selection of learning algorithms. As mentioned earlier, we have tested a range of classifiers trying to get a feeling of the best accuracy we can achieve for this specific problem. Based on empirical results, decision tree learners have the highest accuracy in most cases among all learners considered. The accuracy of rule learners is very close to that of decision tree learners. Since the outputs of rule learners are usually very compact, which is a crucial factor to consider in performing classifications on motes, all the experiments in Section 5 use rule learners.

Selection of features. The impact of feature selection to learning accuracy, memory footprint and FP rate of class c (bad) is demonstrated in Table 3. In particular, it compares 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 just one. This supports our motivation to study more features.

Hardware dependency. Although our supervised classification process requires a manual method to label link quality, it is not dependent on any particular metric, such as LQI available only on

```
rssi <= 212
|   depth <= 5
|   |   rssi <= 211: bad (320.0/37.0)
|   |   rssi > 211: good (79.0/34.0)
|   depth > 5: bad (425.0/31.0)
rssi > 212
|   rssi <= 223
|   |   cla <= 116
|   |   |   depth <= 3: good (352.0/82.0)
|   |   |   depth > 3
|   |   |   |   depth <= 4
|   |   |   |   |   rssi <= 220: bad (49.0/1.0)
|   |   |   |   |   rssi > 220
|   |   |   |   |   |   cla <= 8: good (69.0/29.0)
|   |   |   |   |   |   cla > 8: bad (14.0/4.0)
|   |   |   |   depth > 4
|   |   |   |   |   depth <= 6
|   |   |   |   |   |   rssi <= 216
|   |   |   |   |   |   |   depth <= 5: good (198.0/71.0)
|   |   |   |   |   |   |   depth > 5
|   |   |   |   |   |   |   |   rssi <= 214: bad (8.0/1.0)
|   |   |   |   |   |   |   |   rssi > 214
|   |   |   |   |   |   |   |   |   sendbuf <= 0
|   |   |   |   |   |   |   |   |   |   cla <= 21: bad (29.0/13.0)
|   |   |   |   |   |   |   |   |   |   cla > 21: good (2.0)
|   |   |   |   |   |   |   |   |   sendbuf > 0: good (2.0)
|   |   |   |   |   |   rssi > 216: good (178.0/34.0)
|   |   |   |   |   depth > 6
|   |   |   |   |   |   rssi <= 219
|   |   |   |   |   |   |   rssi <= 215: good (157.0/55.0)
|   |   |   |   |   |   |   rssi > 215
|   |   |   |   |   |   |   |   depth <= 7
|   |   |   |   |   |   |   |   |   rssi <= 217: bad (129.0/29.0)
|   |   |   |   |   |   |   |   |   rssi > 217
|   |   |   |   |   |   |   |   |   |   cla <= 0: good (20.0/6.0)
|   |   |   |   |   |   |   |   |   |   cla > 0: bad (12.0/3.0)
|   |   |   |   |   |   |   |   depth > 7
|   |   |   |   |   |   |   |   |   rssi <= 217: good (37.0/17.0)
|   |   |   |   |   |   |   |   |   rssi > 217
|   |   |   |   |   |   |   |   |   |   cla <= 0: bad (21.0/3.0)
|   |   |   |   |   |   |   |   |   |   cla > 0: good (2.0)
|   |   |   |   |   |   rssi > 219
|   |   |   |   |   |   |   depth <= 7
|   |   |   |   |   |   |   |   cla <= 3: good (102.0/35.0)
|   |   |   |   |   |   |   |   cla > 3: bad (30.0/12.0)
|   |   |   |   |   |   |   depth > 7: good (85.0/17.0)
|   |   cla > 116: good (62.0/8.0)
|   rssi > 223: good (275.0/38.0)
```

Figure 3: A sample decision tree output from Weka using a binary model for labeling. Each line represents one conditional branch in the tree. The pair of number (m/n) behind the label on each line means that there are a total of m instances that reach that leaf, of which n is classified incorrectly.

	7-feature	1-feature (RSSI)	1-feature (pSend)
accuracy	80.8%	70.5%	69.3%
overhead	16 rules	4 rules	20 rules
bad FP rate	4.0%	3.9%	4.1%

Table 3: Impact of feature selection.

802.15.4 radios. Any other available metric, if indicative of link quality, can also be used for labeling.

4. CASE STUDY

In this section, we present a case study to illustrate how supervised learning techniques can be leveraged to improve the performance of link-quality aware collection routing protocols in congested wireless sensor networks.

MintRoute is a collection routing protocol that uses ETX to construct routing topologies. As shown in Figure 1, MintRoute fails to find parents in congested networks, using snooping-based link quality estimation. However, if a parent can be identified based on other available information regarding link delivery capability, routing can be resumed and orphan nodes will be salvaged. 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

```
// 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
  }
}
```

Figure 4: Pseudo-code of MetricMap.

arrival 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. The pseudo-code of MetricMap is listed in Figure 4, with the function `classify()` implementing the second component and the rest functions implementing the first component.

5. TESTBED EVALUATION

To evaluate the efficiency of our technique in real-world sensor network application settings, this section presents our results of experiments implementing the MetricMap prototype in TinyOS and deployed over a real-world wireless sensor network testbed.

The testbed (MoteLab [19]) consists of 30 MicaZ motes across multiple offices in the Harvard Computer Science Building. Motes are connected to an Ethernet used for logging and re-programming. Each MicaZ mote has an 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 Chipcon CC2420, a single-chip IEEE 802.15.4 compliant Radio Frequency transceiver operating at 2.4 GHz and capable of transmitting at 250 kbps. The packet size for the experiments is 29 bytes.

5.1 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 [12] is used to measure the variability of performance across all source nodes. 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 ith 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 measuring the **memory footprint** of each protocol.

Our experiment consists of two phases: the offline training phase, which takes multiple hours for collecting training samples and processing the learning task using Weka; and the online optimization phase that uses the induction rules learned in the training phase to estimate link quality when traditional approaches fail. The training is conducted only once and the output (a classifier) is reused for all experiments with MetricMap. Each test lasts 15 minutes.

Due to uncontrollable factors in the testbed, especially the temporal variability of links, experimental results may be very different across runs at different time. To reduce the impact from such uncontrollable factors, we run MintRoute immediately followed by MetricMap or vice versa. We run such pairs of experiment 5 times and every experiment is independent with respect to each other. Such a design allows us to minimize influences from factors other than the protocol 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.

5.2 Results

Performance and Overhead. Figure 5 compares the data delivery rate between MetricMap and MintRoute. Our approach 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 6 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 7 compares the fairness index of packet delivery. It demonstrates that our approach is much more 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 toward 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 relative 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 normal communication, which actually exacerbates the problem by injecting extra traffic into the network.

Our classifier can mitigate such problems by discerning meaningful link information without imposing any additional traffic. Once the routing structure is restored, data collection can be resumed immediately. Therefore, MetricMap allows more nodes to deliver their data to the sink, which results in a higher fairness index. In contrast, with MintRoute, a few nodes deliver most of their packets while the rest have only a small fraction of their packets delivered.

In summary, MetricMap addresses the high data rate challenge with a different perspective, compared to congestion control mechanisms [23, 24, 11]. Therefore, our approach is orthogonal to theirs and combining them will potentially achieve further performance improvement.

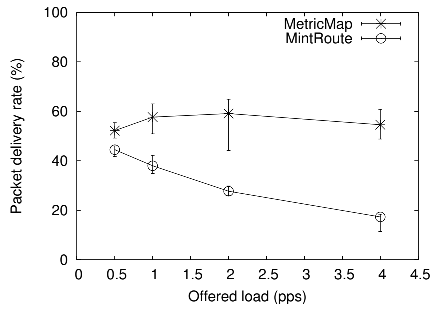


Figure 5: Average success rate versus per-sensor load using a periodic workload.

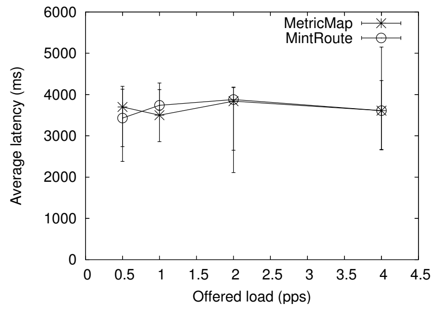


Figure 6: Average packet latency versus per-sensor load using a periodic workload.

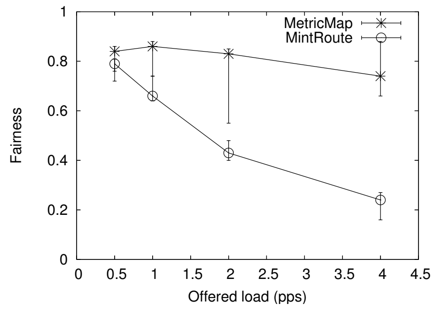


Figure 7: Fairness versus per-sensor load using a periodic workload.

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

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

Table 4: Code and memory usage comparisons of MintRoute and MetricMap on MicaZ. RAM is memory usage in bytes and ROM is program size in bytes.

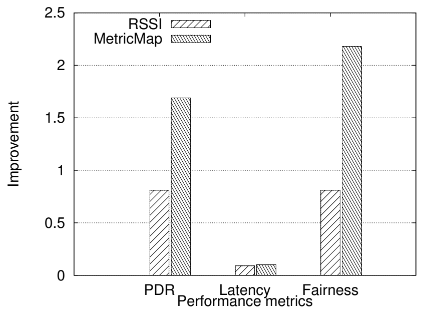


Figure 8: Performance improvement comparison with heuristics-based approach.

shown in Table 4. Table 4 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 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*. RSSI uses the RSSI values of received packets over a link as the only indication of its quality. If the recent received packets have higher RSSI values compared to other links, the protocol will assign a higher quality value to this link than other ones. Other than that, RSSI is the same as MintRoute. Thus, RSSI does not take into account of any factors other than packet RSSI values and is one such protocol that makes its estimation using heuristics.

Figure 8 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 $\frac{p_{RSSI} - p_{MintRoute}}{p_{MintRoute}}$. 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.

6. 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 relevant work in this aspect within sensor networks. We also briefly reviews 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 an significant research focus [13, 5, 4]. Koksal et al. [13] develop new metrics that capture both long-term link quality and short-term variability of the radio channel. Cerpa et al. [5, 4] also study statistical temporal properties of links in low power wireless communications, including both short-term and long-term temporal properties. Such information is then used to develop their link cost model.

All the aforementioned approaches use models to select their metrics. The performance of their metrics depends heavily on their model accuracies, which need much trial-and-error tuning and expert knowledge. Our approach, on the other hand, passively collects features that are readily available 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 is much literature on applying machine learning to different areas of research, and most recently system-related problems, such as compiler optimization [3], system performance diagnosis [7], fault localization in Internet services [6], and software bug isolation [16].

Machine learning has also been used in other areas of sensor networks. Guestrin et al. [10] propose to use 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. [14] study sensor placements using probabilistic models that take both data quality and communication costs into account. Our approach, however, focuses on routing optimizations in the network stack.

7. CONCLUSIONS AND FUTURE WORK

This paper presents a supervised learning framework that can be used to produce useful information automatically and help to make informed decisions in sensor networks. As a case study, we investigate the link quality estimation problem, which is casted as a classification problem using our framework. Results on a real-world sensor network testbed show that our technique can achieve significant performance improvement over existing approaches.

Other than performance improvement, the complexity ramifications of this work compared to existing approaches using heuristics, are very encouraging. Furthermore, our proposed framework is general enough to be applied to other problems that could benefit from such information discovering. Overall, it provides a new direction toward routing optimizations in planning and deploying real-world sensor networks.

In the future, we plan to study the potential of using unsupervised learning techniques to reduce the cost of labeling. Another important area of future work is to investigate the feasibility of online, incremental training in a distributed fashion. This approach has the benefits of being able to quickly adapt to network dynamics if the network varies significantly over time and space. However, it will involve a different set of tradeoffs between resource usage and performance.

8. REFERENCES

[1] *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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

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

[9] 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.

[10] 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.

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

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

[13] 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*, 2005.

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

[15] 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.

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

[17] Mistlab. http://mistlab.csail.mit.edu/.

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

[19] Motelab. http://motelab.eecs.harvard.edu/.

[20] 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 EmNetS*, 2005.

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

[22] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

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

[24] 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.

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

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