2009 International Conference on Computational Science and Engineering

MALADY: A Machine Learning-based Autonomous Decision-Making System for Sensor Networks

Sudha Krishnamurthy
United Technologies Research Center
East Hartford, CT

Geethapriya Thamilarasu
Dept. of Computer Science & Engineering
University of Buffalo

Christian Bauckhage
University of Bonn
Bonn, Germany

***Abstract*—As the capabilities of sensor networks evolve, we need to address the challenges that will help in shifting the perception of sensor networks as being merely a data-gathering network to that of a network that is capable of learning and making decisions autonomously. This shift in intelligence from the edge to the nodes in the network is particularly relevant in unattended sensor deployments where there is no continuous access to a remote base station. In this paper, we propose an architecture, called MALADY, which uses a machine learning approach to enable a network of embedded sensor nodes to use the data that they have gathered to learn and make decisions in real-time within the network and thereby, become autonomous. MALADY supports supervised as well as unsupervised learning algorithms. Our implementation of the algorithms introduces some practical optimizations, in order to make them viable for nodes with limited resources. Our experimental results based on an implementation on the MicaZ mote/TinyOS platform show that the supervised learning technique based on linear discriminant analysis has a higher learning complexity, but allows a sensor node to learn about the data correlations robustly and make decisions accurately, after learning from only a few samples. In comparison, the unsupervised learning technique based on clustering has a low overhead, but requires more learning samples to achieve a high detection accuracy.**

I. Introduction

One of the characteristics that distinguishes sensor networks from other types of networks is that they are capable of being embedded within physical environments, where they can function in an unattended manner. These sensor nodes can sense and gather different types of information from the environments in which they are embedded and use that to learn and make decisions in real-time. This ability is particularly useful in environments where it is not always possible to transmit the data gathered by the network to powerful base stations for making decisions, because of intermittent access to the remote base stations. In this paper, we address the problem of how we can enable a network of embedded sensor nodes with constrained resources deployed in such intermittently connected environments to become autonomous. Our approach enables the nodes that are deployed within the network to use machine learning techniques for autonomous decision making. This poses some challenges that need to be addressed in the context of sensor networks.

This work was done when Sudha Krishnamurthy and Christian Bauckhage were affiliated with Deutsche Telekom Laboratories in Berlin, Germany and Geethapriya Thamilarasu was a summer intern there.

First, some of the machine learning techniques while being effective, also tend to be computationally intensive and may require a long learning period [3]. Hence, given the limited computational and storage resources of sensor networks, one of the challenges is to select the learning and inferencing techniques that are effective in a variety of scenarios and are at the same time practical to implement on nodes with constrained resources. Second, the different types of data gathered by a sensor node may be correlated and the information may not always be precise. Hence, the learning techniques should be able to exploit these data correlations and be effective even when there is some degree of imprecision in the data. Third, in some cases, example patterns (known as labeled samples) may be available for guiding the learning process, whereas in other cases, such guidance may not be available, since the sensor network applications are still evolving. Hence, the learning and inferencing architecture for a sensor network has to be adaptive and support both supervised and unsupervised learning.

A. Contributions

Our work targets autonomous sensor environments, where learning and decision-making is distributed within the network. We explore whether resource-constrained sensor nodes are capable of learning and inferencing using techniques that are more complex than the application-specific, rule-based learning that has been used previously in sensor networks. Instead of developing point solutions for specific tasks, we propose a generic learning and inferencing architecture, called MALADY, that can be used by the nodes in a sensor network to make decisions in the context of different applications, by gathering inputs that are relevant to the application in real-time. Since the different types of attributes collected by the sensor nodes in an application are often correlated, the learning techniques in MALADY have been chosen to exploit these data correlations effectively.

Learning may be done through supervision when example patterns (known as labeled samples) are available for guiding the learning process. MALADY supports supervised learning through the use of the linear discriminant analysis technique [3], [4], [5]. However, since the sensor network applications are still evolving and the environments are unpredictable, for certain tasks, the network may need to learn without any guidance, by using unlabeled patterns. Hence, MALADY also supports unsupervised learning through the use of the fixed-width clustering algorithm [6]. Data collection, learning, and inferencing in MALADY is a continuous process that happens in real-time. By interspersing the learning and inferencing phases, MALADY enables the network to adapt its learning and inferencing regularly, based on new data gathered by the individual nodes. MALADY has been implemented in TinyOS and runs as a library that can be linked to any TinyOS application. Our evaluation demonstrates that the sensor nodes, despite their limited resources, can learn and make decisions effectively in real-time using supervised as well as unsupervised techniques.

B. Related Work

We now summarize some of the related work that has applied learning approaches for making decisions at different layers of a sensor network. RL-MAC [7] is an adaptive MAC layer protocol, in which the nodes make use of reinforcement learning to learn the dynamics of the MAC layer and infer the state of neighbors, in order to decide when to schedule their transmissions. Learning approaches have been used to design adaptive routing algorithms in sensor networks by applying rule-based learning techniques [8] and reinforcement learning approaches [9]. The scheme proposed in [2] applies supervised rule-based approach for offline learning at the sink node using the data collected in the network and applies that knowledge to make informed routing decisions online in the network. In contrast, our work targets autonomous sensor environments, where both learning and decision-making needs to happen within the network.

Machine learning has also been used for different application-level tasks in sensor networks. In [1], a supervised learning technique based on support vector machines (SVM) [5] has been proposed for performing classification using wireless sensor networks. The classification is done using a distributed learning approach. Each clusterhead incrementally estimates its local SVM by using as inputs the SVM estimated by its neighboring cluster and the samples recorded by the sensor nodes in its own cluster. In [10], a method based on unsupervised cluster-based learning techniques has been used to detect routing anomalies in sensor networks. Most of the efforts mentioned above have, however, been primarily evaluated through simulation.

In Section II, we describe the MALADY architecture and the machine learning techniques it employs. In Section III, as an example application, we illustrate the use of MALADY in detecting some of the anomalies that are relevant to sensor networks. In Section IV, we present experimental results that evaluate the performance of the learning algorithms for anomaly detection, based on an implementation of MALADY on MicaZ motes running TinyOS. In Section V, we present our conclusions.

II. Description of the MALADY Architecture

We now summarize the key design principles that form the basis of the MALADY architecture. These principles were derived from typical characteristics of sensor networks and by considering some of the decisions that a sensor network

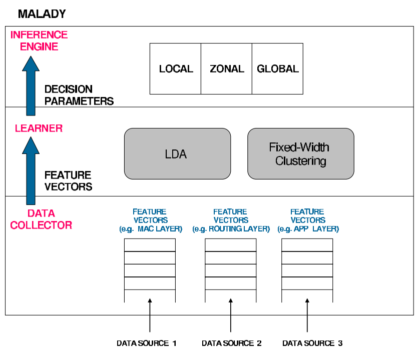


Fig. 1. Overview of the MALADY architecture. The buffers in the data collector are used to store different types of features that are collected.

can learn to make. First, sensor nodes sleep and wake up periodically and no single node has the resources to capture and store all of the statistics. So the data collection and learning in MALADY is distributed among the nodes in the network. Second, the different types of features collected by a node may be correlated and these correlations need to be considered while making decisions. Hence, instead of learning from each feature individually, the algorithms we have implemented in MALADY allow a node to learn the correlations by grouping the features together. Third, in order to serve as a generic learning architecture for different applications, MALADY runs as a library that can be linked to any application and exposes simple abstractions that an application can use to input application-specific parameters (e.g. number of learning samples). MALADY is adaptive and supports both supervised and unsupervised learning. When labeled samples are available, the supervised linear discriminant analysis (LDA) learning technique [3], [4], [5] is used, whereas the unsupervised fixed-width clustering algorithm [6] is used for learning when example patterns are unavailable. Finally, the MALADY architecture is extensible and provides common abstractions that may be used to implement learning algorithms, in addition to those that we have implemented. The MALADY system runs locally on each node and is made up of the data collector, learning, and inference modules, as shown in Figure 1. We now describe each of these modules.

A. Data Collector Module

Data collection in MALADY is decentralized. Whenever a sensor node is awake, it periodically measures the features that serve as inputs to the learning and decision processes. The features that are collected depend on the type of decisions that are being made in the network and can be specified by the application. MALADY allows cross-layered information to be gathered, as shown in Figure 1. The data collector module periodically pulls the different measurements with a frequency that is specified by the application and stores them in buffers. Since different feature values may have different ranges and MALADY is intended as a generic learning and inferencing architecture for a variety of tasks, the data collector normalizes the feature values by collecting K measurements of each feature during a given time interval and constructs a feature vector $\vec{F}$, so that each feature value in the vector lies in the range [0.0, 1.0]. We illustrate this normalization with an example later in Section IV. After normalization, the new feature vector $\vec{F}$ constructed by the data collector is passed up as input to the learning module.

B. Learning Module

During the learning phase, the learning module enables a node in the network to learn from the set of data samples that it collects. MALADY supports both supervised and unsupervised learning techniques. When the samples correspond to patterns that are already known to the network, the data collector labels the samples to guide the learning and supplies them as input to the learning module. When the sample patterns correspond to previously unknown patterns, the learning samples are unlabeled. The machine learning literature offers several choices for the learning algorithms. Algorithms that require prolonged learning and involve highly complex computations are not feasible to implement on an embedded environment with limited computational, storage, and energy resources. However, these sensor nodes are capable of solving linear systems of equations. We now summarize some of the learning techniques that we considered and then present the rationale for the technique we chose for MALADY.

Decision tree learning is one of the supervised learning mechanisms that is widely used for solving classification problems [3]. Although inferencing using a decision tree is computationally simple, the trees can easily grow to a large size as new rules emerge and this consumes significant memory, which is a limited resource in sensor nodes. Rule-based learning is another supervised technique which uses IF-THEN rules for making decisions. The accuracy of detection depends on formulating well-formed rules and it may not always be possible to completely automate the process of generating the rules and optimize the rule processing, as new rules emerge. Support vector machine (SVM) (cf. e.g. [5]) is another kind of supervised learning technique that generates a separating hyperplane from the labeled training vectors, which is then used for binary classification. SVMs generally have a short training time, but involve solving quadratic optimization problems which we found too computationally intensive to implement on the embedded sensor platform.

Compared to the SVM approach, the linear discriminant analysis (LDA) approach [11], [12], which is also a supervised learning technique, is considered a lightweight method. LDA is known to have robust and reliable performance in most practical applications [12]. Unlike rule-based learning, which requires different rules to be provided for different applications, the LDA algorithm can be used for learning from different feature sets. Hence, the supervised learning technique in MALADY is based on LDA. In addition to supervised learning, MALADY supports unsupervised learning, which is based on the fixed-width clustering algorithm [6], [10]. We now describe the implementation of the LDA and unsupervised clustering techniques in MALADY.

Algorithm 1 : LDA Implementation in MALADY

```
1: Learning phase: input:: $< \{F_i, C_i\} | C_i$ is the class of feature vector $F_i >$
2: for each vector $F_i$ of length $> 3$ do
3:    Partition $F_i$ into disjoint, sub-vectors $\{F_{i1}, F_{i2}, \ldots, F_{ik}\}$, each having a
      length of at most 3;
4: end for
5: for each class $C_i$ do
6:    From the learning samples, compute the class mean sub-vectors $M_{ik}$ for class
      $C_i$;
7: end for
8: Compute the global mean feature vector $G$ from the class mean sub-vectors;
9: for each class $C_i$ do
10:    for each sample sub-vector $F_{ik} \in C_i$ do
11:       Determine the intra-class scatter by computing the deviation of $F_{ik}$ from
          the global mean feature vector $G$;
12:    end for
13: end for
14: for each class $C_i$ do
15:    Compute the covariance matrices $V_{ik}$ from the $k$ mean sub-vectors of class $C_i$;
       Each $V_{ik}$ is a matrix of maximum dimension 3.
16: end for
17: From the $V_{ik}$ matrices, compute the group covariance matrices $R_k$ across all classes
    for each subvector;
18: Compute the inverse covariance matrix $R_k^{-1}$ of $R_k$; Each $R_k$ is a square matrix
    of maximum dimension 3.
19: for each class $C_i$ do
20:    $Z_{ik} = M_{ik} \star R_k^{-1}$;
21:    Compute the Mahalanobis distances $l_{ik}$ for class $C_i$ as follows, where $M_{ik}^T$ is
       the transpose of the vector $M_{ik}$
       $l_{ik} = 0.5 \star Z_{ik} \star M_{ik}^T$
22: end for
23: Decision phase: input:: $<$ test vector $W >$
24: for each class $C_i$ do
25:    Compute the partial linear discriminant function $d_{ik}$ with respect to test vector
       $W$ for class $C_i$ using the following equation:
       $d_{ik} = Z_{ik} \star W^T + l_{ik}$
26:    Compute the total linear discriminant function $d_i$ with respect to test vector $W$
       for class $C_i$ using the following equation:
       $d_i = d_{i1} + d_{i2} + \ldots + d_{ik}$
27: end for
28: Test vector $W$ belongs to the class $j$ that results in the maximum value of the
    discriminant function $d_j$.
```

1) Linear Discriminant Analysis: Discriminant analysis is a statistical technique that is used to classify objects into mutually exclusive and exhaustive groups based on a set of measurable object's features. The basic steps involved in the LDA algorithm are outlined in [4]. We have introduced some modifications that make it practical to implement the LDA technique on nodes with limited computational and storage resources. Our implementation of the discriminant analysis technique in MALADY can be used to detect and classify an observation into one of multiple known classes and is outlined in Algorithm 1. Given a labeled set of N observations $\{(\vec{F_i}, C_i) | i = 1, \ldots, N\}$, where the vector $\vec{F_i} \in \mathbb{R}^n$ is a feature vector consisting of n features and C_i denotes the label of the class to which $\vec{F_i}$ belongs, LDA tries to maximize the separation between the known classes during the learning phase, so that there is minimal ambiguity in detecting the class to which a test pattern belongs.

The LDA learning process performs several operations, including the computation of the class mean for each class, the group covariance matrix, the inverse of the group covariance matrix, and finally, the distance function for each class. One of the challenges in using the LDA technique in an embedded sensor platform with limited resources (like the motes) is the computation of the inverse matrix of dimension n, where n is the size of the feature vector. For smaller dimensions

($n \leq 3$), the inverse matrix can be easily computed. However, computing the inverse of a matrix of larger dimensions, can be computationally intensive on embedded sensor platforms. So we have modified the LDA algorithm to use a divide and conquer approach. In the first step (Lines 2-4 of Algorithm 1), we reduce the dimensionality of the original feature vector by partitioning each feature vector $\vec{F_i}$ of a class i into smaller, disjoint, sub-vectors $\{F_{i1}, F_{i2}, \ldots, F_{ik}\}$, such that all of the feature sub-vectors have a dimension of at most 3. For example, if the original feature vector of a class i has 5 features, $\vec{F_i} = [f_1, f_2, f_3, f_4, f_5]$, then we partition $\vec{F_i}$ into 2 sub-vectors, $\vec{F_{i1}} = [f_1, f_2, f_3]$ and $\vec{F_{i2}} = [f_4, f_5]$, where $\vec{F_i} = \vec{F_{i1}} \cup \vec{F_{i2}}$. In Line 17, as a result of partitioning the feature vectors of each class into k sub-vectors, each of maximum length 3, we obtain k corresponding group covariance matrices. The nodes can then easily compute the inverses of these group covariance matrices (Line 18), because they all have a dimension of at most 3. At the end of a learning phase (Lines 19-22), the mean feature vectors $\{M_{i1}, M_{i2}, \ldots, M_{ik}\}$ for each class i and the k inverse covariance matrices are used to compute a set of distance measures $\{l_{i1}, l_{i2}, \ldots, l_{ik}\}$, where l_{ik} is the Mahalanobis distance corresponding to the feature subvector F_{ik} of class i. The Mahalanobis distance is a way to measure the similarity between two sample sets by taking into account the correlations of the datasets.

During the decision phase, the test vector W has to be compared with each class to find the class that W belongs to. To do that, the inverse covariance matrices and Mahalanobis distance values obtained during the learning phase are used to compute the partial discriminant value for each sub-vector F_{ik} of a class C_i (Line 25 of Algorithm 1). These partial discriminant values are then summed up to compute the total discriminant value of class C_i as $d_i = d_{i1} + d_{i2} + \ldots + d_{ik}$ (Line 26 of Algorithm 1). The test vector W is then assigned to the class j that yields the maximum value of the discriminant function for that test vector, which indicates that the samples in j are the closest match for the test vector W. We present the experimental evaluation of the LDA implementation in Section IV.

2) Fixed-Width Clustering: The unsupervised fixed-width clustering algorithm we have implemented is presented in Algorithm 2 and like the LDA implementation, incorporates some practical optimizations that make it easy for computation on nodes with limited computational and storage resources.

During the learning phase, each new feature vector $\vec{F_k} = [f_1, \ldots, f_n]$ consisting of n features is compared with the mean feature vector $\vec{M}$ of each of the currently existing clusters. The comparison is based on the Manhattan distance measure between two vectors, which indicates how similar the new sample is to the average samples in a cluster. The new sample is included in the cluster that results in the smallest distance d such that $d \leq r$, where r is a specified threshold value (Lines 7-8). The mean feature vector of that cluster is then updated. We currently use a fixed threshold for all of the clusters and hence, the algorithm is called fixed-width clustering. If none of the existing clusters yields a

Algorithm 2 : Fixed-Width Clustering Implementation in MALADY

```
 1: Learning phase:
    input:: $<$ Unlabeled learning feature vectors($F_k$), fixed width($r$) $>$
 2: for each cluster $C_i$ do
 3:     Compute the Manhattan distance $d_i$ between learning vector $F_k$ and the mean
        feature vector $M_i$ of cluster $C_i$;
 4: end for
 5: Determine the cluster $j$ that results in the minimum distance $d_j$
 6: if ($d_j \leq r$) then
 7:     Assign $F_i$ to cluster $j$
 8:     Update the mean feature vector $M_i$ of cluster $C_j$;
 9: else
10:     Create a new class with $F_k$ as the initial mean feature vector
11: end if
12: Decision phase: input:: $<$ test vector $W$ $>$
13: for each cluster $C_i$ do
14:     Compute the Manhattan distance $d_i$ between test vector $W$ and the mean feature
        vector $M_i$ of cluster $C_i$;
15: end for
16: Determine the cluster $j$ that results in the minimum distance $d_j$
17: if ($d_j \leq r$) then
18:     Test vector $W$ belongs to the cluster $j$
19: else
20:     No matching cluster found
21: end if
```

distance value that is lower than the threshold r, it indicates that none of the existing clusters is a close match for the sample. Therefore, a new cluster is formed with that sample as the initial member (Line 10). During the decision and classification phase (Lines 12-21), the clustering algorithm uses the same distance measure that is used in the learning phase to find the closest matching cluster for the test vector.

C. Inference Module

The inference module makes use of the knowledge acquired during learning for making decisions in real-time. The inference engine in MALADY is organized as shown in Figure 1. Any individual node is allowed to make inferences based on the training it acquired during the learning phase, by using the feature samples it gathered. The decision algorithm employed by the local inference engine depends on the learning technique that is used. If the LDA technique is used for learning, then the decision phase also uses the LDA decision algorithm (lines 23-28 of Algorithm 1). On the other hand, if the clustering learning technique is used, then the local inference engine makes decisions using the clustering technique (lines 12-21 of Algorithm 2).

Since the communication overhead in sensor networks has to be minimized for energy efficiency, local inferencing is the preferred mode in MALADY. However, in sensor networks, since nodes periodically sleep and wake up to conserve energy, in some cases, the information gathered by a single node may only result in partial or approximate inferences. To improve the inferencing, the MALADY architecture has been designed to also support collaborative inferencing, in which nodes in a neighborhood (e.g. within a zone or grid) can collaborate with their neighbors or manager and reach a consensus using majority voting. The implementation of collaborative inferencing on the sensor nodes is ongoing work. Hence, we focus on local inferencing in this paper.

TABLE I
FEATURE SELECTION FOR ANOMALY DETECTION

Anomaly	**Detection features**	**Feature collection layer**
Physical jamming	Carrier sensing count Packet send ratio	Physical layer Link Layer
Collisions	Retransmission count	Link layer
Selective forwarding	Carrier sensing count Packet send ratio Retransmission count Packet drop ratio	Physical layer Link layer Routing layer
Misdirection	Misdirection count	Routing layer

III. APPLICATION: ANOMALY DETECTION

MALADY has been designed with the goal of supporting learning and decision-making in different sensor network applications. However, since the primary motivation of our work is to enable a sensor network to become autonomous, especially in unattended environments, we now illustrate the use of MALADY for a problem that is particularly relevant in the context of unattended sensor deployments. The unattended nature of sensor networks, combined with their proximity to open physical environments, makes them vulnerable to several anomalies, some of which lead to unnecessary energy depletion in the network. Since there is no continuous access to an external base station that can monitor the anomalies in real-time in such environments, the nodes in the network must be capable of learning about the anomalies and detecting them, so that the sensor network can become responsive to those anomalies. Some of the anomalies may be known and the network can learn about such anomalies with the help of labeled samples using supervised learning. On the other hand, the deployed sensor network may not be previously exposed to some of the anomalies. In such cases, the network can employ unsupervised learning to learn from the learning samples that the nodes collect. Furthermore, a node may need to learn about the correlations among multiple features, in order to detect different anomalies. The data collector module in MALADY allows information to be collected from across different layers. Moreover, the supervised and unsupervised learning algorithms in MALADY allow different features to be grouped together and take into account the correlations among the features while learning. Therefore, they can be used to detect different types of anomalies. We illustrate the use of MALADY in detecting anomalies that are likely to lead to energy exhaustion in an unattended sensor network, some of which are listed in Table I. We first briefly describe the anomalies listed in Table I and then in the next section, evaluate the performance of MALADY in detecting those anomalies.

Physical jamming: Physical jamming occurs when an anomalous node deliberately transmits RF signals that interfere with the radio reception of the wireless communication channel [15]. As a result, a normal node in the neighborhood that is attempting to transmit is likely to sense the channel to be busy, forcing the node to backoff for sometime, increment its carrier sensing count, and attempt its transmission again. Continuous transmission of jamming signals depletes the energy resources of a node that repeatedly attempts to retransmit. To detect this anomaly, each node monitors its own carrier sensing count and packet send ratio. The carrier sensing count is a physical layer attribute that determines the number of times a node senses the channel to be busy and has to backoff, before transmitting a packet. The packet send ratio (PSR) is a link layer attribute that measures the fraction of packets that are successfully sent out at the link layer of a node during an interval t. The probability of a higher than normal carrier sensing count along with a lower than normal PSR over a period of time is used to detect the presence of the physical jamming anomaly.

Collisions: Instead of continuously transmitting high frequency signals, as in the case of physical jamming, an anomalous node may either choose to transmit periodic jamming signals or reactively begin a transmission only when it overhears another transmission in the channel. This type of intelligent jamming anomaly results in deliberate collisions with normal transmissions. To generate this anomaly, the jammer in our sensor network causes collisions with the DATA packets, forcing the sender to retransmit and increment its retransmission count, which is recorded as a link layer attribute. The probability of a higher than normal retransmission count is used to detect the presence of an anomalous node that is deliberately causing collisions.

Misdirection: The misdirection anomaly occurs when an anomalous node forwards packets to a wrong destination D. As a result, D wastes its energy by receiving packets not originally intended for it. In order to detect this anomaly, each sender node S snoops on the channel after forwarding a packet to its next-hop node N. The next-hop node in many sensor network routing protocols is the cluster manager or the parent node in the routing tree. If the next-hop node N, forwards the packet by changing the original destination of the packet, then the node S increments the misdirection packet count for node N. The node S can then compute the probability that the misdirected packet ratio for its next-hop node N (defined as $\frac{\text{misdirection count of N as measured by S}}{\text{number of packets forwarded to N by S}}$) is above a certain threshold and use that to detect misdirections.

Selective Forwarding: When a node forwards only a subset of the packets that it receives, it exhibits the selective forwarding anomaly [13]. Different variants of selective forwarding are possible. For example, a node may select packets for forwarding randomly, or choose to forward only those packets that either originate at a specific source or those that are intended for a specific destination. In the extreme case, when the node drops all of the packets that it receives, it results in the blackhole attack. A node may drop packets on account of multiple reasons. The node may be anomalous and therefore, it may intentionally drop the packets. Alternatively, a node may be forced to drop packets due to buffer overflow at the lower layers, resulting from either continuous channel contention or excessive collisions. Hence, in order to detect and distinguish between these different causes, the learning and inferencing algorithms need to correlate the information collected from different layers.

The carrier sensing count measured by a node S at the physical layer, the PSR and retransmission count values measured by S at the link layer, and the packet drop ratio (PDR) measured by S for its next-hop node N over a period of time t at the routing layer (defined as $\frac{\text{number of packets dropped by N as measured by S during t}}{\text{number of packets forwarded to N by S during t}}$) can all be correlated to detect the selective forwarding anomaly, as follows. If the PSR of a legitimate node S is low and its carrier sensing count is high, then that may indicate a jamming condition. In such a case, if N consistently has a high PDR, then that may be caused by channel contention. On the other hand, if the retransmission count is high, then a high PDR for N may be the result of excessive collisions in the neighborhood. If neither of these two anomalous conditions are true, and S measures the PDR of its next-hop node N to be higher than a certain threshold (or equal to 1) over a period of time, then it is highly likely that N is exhibiting the selective forwarding anomaly.

IV. Performance Evaluation

We have implemented the MALADY architecture in TinyOS and tested it on an ad-hoc network of MicaZ motes. In this section, we present results that compare the overhead and effectiveness of the algorithms implemented in MALADY for detecting the anomalies listed in Table I. The MicaZ mote has 128 KB of programmable flash memory for storing code and 4 KB of EEPROM for storing runtime data. Our implementation statically allocates the memory for the buffers that store the learning samples and for the matrices computed in the LDA process, because the use of dynamic memory allocation on the physical motes is discouraged in TinyOS. When only the clustering algorithm is active, the library code along with the anomaly detection application consumes 22036 bytes of flash memory and 1882 bytes of data memory. When only the LDA algorithm is used, the corresponding footprint is 27906 bytes of flash memory and 2827 bytes of data memory. When both the LDA and clustering algorithms are active, they together consume 31342 bytes of flash and 3500 bytes of data memory.

Each node in the network monitors features that are relevant for detecting the anomalies listed in Table I. At any given time, only one of the nodes in the network exhibits anomalous behavior. The data collector in each node j collects K measurements of each feature during a given time interval and then normalizes the feature values f_i, as follows:

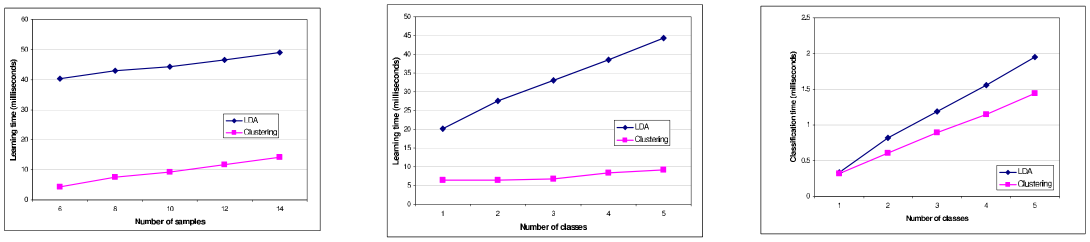
1) f_1 : The fraction of measurements (among K measurements) that have a carrier sense count value greater than a threshold T_1. So if $T_1 = 8$, $K = 10$, and 5 out of the K observations have carrier sense count values higher than 8, then the value of f_1 is 0.5.
2) f_2 : The probability that the packet send ratio is less than a threshold T_2, among a set of K observations.
3) f_3 : The probability that the data retransmission count value is greater than a threshold T_3, among a set of K observations.
4) f_4 : The ratio of the total number of packets forwarded by node j that were dropped by its next-hop node (parent) to the total number of packets that were forwarded by node j to its parent during the K observations.
5) f_5 : The ratio of the total number of packets forwarded by node j that were misdirected by its parent to the total number of packets that were forwarded by node j to its parent during the K observations.

In our implementation, $K = 10$ and we chose suitable values for the different thresholds. The collector module computes the feature probabilities enumerated above for every K observations and constructs a learning feature vector $\vec{F} = [f_1, f_2, f_3, f_4, f_5]$. Using the optimization mentioned in Section II-B, we reduce the dimensionality of the original feature vector by splitting each feature vector $\vec{F_k}$ consisting of the 5 features into two sub-vectors, $F_k^{\vec{LL}} = [f_1, f_2, f_3]$ and $F_k^{\vec{HL}} = [f_4, f_5]$. f_1, f_2, and f_3 are the feature probabilities related to the carrier sensing count, packet send ratio, and data retransmission count and they correspond to the MAC and lower layer features. $F_k^{\vec{HL}}$ is a higher-layer feature vector containing the features of the layers above MAC, where f_4 and f_5 are the feature probabilities related to the packet drop count and misdirection count.

A. Learning and Inferencing Complexity

We now present the results of the experiments we conducted on the motes to study the factors affecting the learning and inferencing time of the LDA and clustering algorithms. Initially, the network is in a state in which it is not exposed to any anomalies and every node collects learning samples corresponding to NORMAL behavior. Then one of the nodes in the network begins to exhibit different anomalies: JAMMING, COLLISION, SELECTIVE FORWARDING, and MISDIRECTION, for a given period of time. Only one type of anomaly is present at any given time and the nodes near the anomalous node collect the learning samples corresponding to that anomaly. The data collector of each (normal) node in the zone, therefore, collects the learning samples for 5 different classes. For evaluating the LDA algorithm, the collector module labels the samples appropriately before sending them to the learning module. The clustering algorithm does not require labeled learning samples and instead, automatically groups the unlabeled samples into fixed-width clusters using Algorithm 2. We used a cluster width of 0.5.

Figure 2(a) shows the time it takes for one of the nodes in the zone to learn from different number of samples. The time was measured using the timing functions in TinyOS. The plots show that the learning time in the case of LDA is significantly higher than in the case of the clustering algorithm, and that there is a marginal increase in the learning time for both algorithms, as the number of samples (i.e. sample size) increases. As the sample size increases from 6 to 14, the learning time using LDA increases from 40.43 to 48.98 milliseconds, while the learning time for unsupervised clustering increases from 4.4 to 14.26 milliseconds. The learning time of LDA is longer compared to clustering, because the LDA learning process in Algorithm 1 performs more computationally intensive operations. On the other hand, the clustering algorithm performs



(a) Impact of sample size on learning time (b) Impact of class size on learning time (c) Impact of class size on inferencing time

Fig. 2. Learning and inferencing time

simple vector operations to compute the distance vectors and the mean vector.

Figure 2(b) shows that the learning time for both the LDA and the unsupervised clustering algorithm is also affected by the number of distinct classes (i.e. class size). To assess the impact of the class size, we use learning samples belonging to the same 5 patterns described above. For each pattern, a node learns from 10 different learning samples that it collects at runtime. From the plots, we see that the learning time for both the algorithms increases with the number of distinct classes. Lines 2-22 of Algorithm 1 show that the number of computations performed by LDA to compute the discriminant function for each class increases linearly with the number of classes, resulting in a linear increase in the learning time. Likewise, in the case of clustering, the number of comparisons that need to be performed to find a matching cluster for each new learning sample increases with the number of distinct clusters (lines 2-11 of Algorithm 2), which explains the marginal increase in the learning time for unsupervised clustering in Figure 2(b).

After each node learns about the 5 different patterns, it is ready to switch to the inferencing phase and detect different types of anomalies. Figure 2(c) shows that like the learning time, the inferencing time also increases with the number of classes in the case of the LDA and the clustering algorithms. However, compared to the learning time, the inferencing time for both the algorithms is much lower and is only of the order of a few milliseconds. Thus, although the learning phases of the LDA and clustering algorithms involve significantly different operations, their decision-making phases involve simple operations on the feature vectors. The low complexity of the decision-making phase using our implementation suggests that it is feasible to use the LDA or the clustering algorithm for runtime decision-making on sensor platforms with constrained resources.

B. *Detection Accuracy*

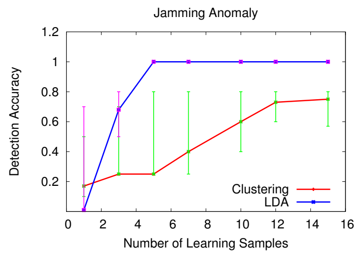
We evaluated the detection accuracy of the LDA and clustering techniques for the anomalies listed in Table I. Figure 3(a) and 3(b) present the results for the jamming and selective forwarding anomalies, based on an evaluation of several hundreds of test vectors. The results show that the LDA algorithm achieves almost 100% detection accuracy after learning from just 5 samples in the case of both types of anomalies. In contrast, even after learning from 15 samples, the clustering algorithm is able to achieve only at most 80% detection accuracy in the case of the jamming anomaly and 90% accuracy in the case of the selective forwarding anomaly. The length of the errorbars show that the performance of the LDA technique is more robust and has little variation, especially after learning from 5 or more samples. In comparison, the clustering algorithm is less robust, although the length of the errorbars show that its robustness also improves with the number of learning samples.

The above results can be explained by looking at some learning samples from both anomaly classes. As explained in Section III, a jamming anomaly is characterized by a high carrier sensing count (f_1) and low PSR value (f_2). A jamming anomaly may cause the packet drop ratio (f_4) of a node within a neighborhood to be high, although that may not always be the case. Thus, using the feature vector description at the beginning of this section, the following two vectors are both valid learning samples for the jamming anomaly:

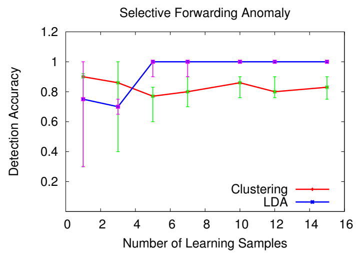
$$\vec{F_1} = [0.8, 0.2, 0.0, 0.2, 0.0, 0.0]$$
$$\vec{F_2} = [0.9, 0.1, 0.0, 0.8, 0.0, 0.0]$$

In the case of LDA, we guide the learning by explicitly assigning the same label to both $\vec{F_1}$ and $\vec{F_2}$. Hence, based on a few different learning samples, a node quickly learns the data correlations through supervision that while the packet drop feature may take on a range of values, it is the carrier sensing count and PSR features that play a more dominant role in detecting the jamming anomaly. In contrast, in the case of unsupervised clustering, when a node is first presented with the above two samples, it assigns the samples to two different clusters, because the Manhattan distance between the two vectors exceeds the clustering width of 0.5. Therefore, more samples are needed, in order to robustly classify such patterns using the clustering technique.

In the case of the selective forwarding or blackhole anomaly, the packet drop feature has a high value, whereas the remaining features have normal values, as explained in Section III. Therefore, detection of this anomaly is less ambiguous compared to the jamming anomaly. This explains why the clustering technique needs fewer learning samples to achieve good detection accuracy in the case of the selective forwarding anomaly, compared to the jamming anomaly. Hence, we conclude that while the clustering technique has good detection accuracy when the feature values are fairly consistent across



(a) Jamming anomaly



(b) Selective forwarding anomaly

Fig. 3. Comparison of the detection accuracy of the LDA and clustering techniques

the different learning samples, it is less robust and has lower accuracy in detecting anomaly patterns that allow a feature to take on a range of different values. On the other hand, the LDA technique is less vulnerable to these ambiguities and performs well in both cases, because it learns through guidance.

Currently, each node monitors the packet drop ratio of its parent (which is its next-hop node), and locally detects whether its parent is exhibiting the selective forwarding anomaly. However, the selective forwarding anomaly has different variants, as explained in Section III. Based on the values collected locally by a node, it is not possible for the node to infer whether its parent is selectively dropping only its packets or whether the parent is exhibiting a blackhole anomaly and dropping all of the packets that are being forwarded to it. In such cases, the detection accuracy of MALADY can be further enhanced through the use of collaborative inferencing, which is part of ongoing work. If all of the zone members that forwarded a packet to their parent N during a given time interval measure the packet drop ratio of N to be 1, then they can collaboratively infer that N is exhibiting the blackhole anomaly. However, if at least one of these zone members measures the packet drop ratio of N to be low or 0, then they are more likely to infer that their parent is selectively forwarding packets.

V. Conclusions

The intersection of machine learning and wireless sensor networks is a rich area for both fundamental and applied research problems. On the one hand, wireless sensor networks can leverage existing machine learning algorithms to become an intelligent and autonomous network. On the other hand, the machine learning community can tap into the vast amounts of correlated data that these sensor nodes gather from real environments, in order to apply or develop learning algorithms. The MALADY architecture demonstrates that reasonably sophisticated learning and inferencing techniques can be adapted in a way that is practical to implement on nodes with constrained resources. Like the anomaly detection application, the data collected in many sensor network applications often exhibit correlations and have some degree of ambiguity and imprecision. Our experimental results show that the nodes can successfully employ both the LDA and clustering algorithms to learn about these data correlations and make reasonably accurate decisions in real-time, even when the data has some degree of imprecision. While the LDA algorithm has a higher learning overhead, it is more robust and enables the nodes to learn about the data correlations more effectively through supervision and make decisions accurately based on just a few training samples. In comparison, the unsupervised fixed-width clustering algorithm has lower overhead, but requires more number of learning samples to achieve a high degree of accuracy, especially when there is imprecision.

Acknowledgments

We thank the management at Deutsche Telekom Labs for supporting this work.

References

[1] K. Flouri, B. Lozano, and P. Tsakalides, "Training an SVM-Based Classifier in Distributed Sensor Networks," in *Proc. of EUSIPCO*, 2006.
[2] Y. Wang, M. Martonosi, and L. Peh, "Supervised Learning in Sensor Networks: New Approaches with Routing, Reliability Optimizations," in *Proc. of SECON*, 2006, pp. 256–265.
[3] N. J. Nilsson, "Introduction to Machine Learning."
[4] K. Teknomo, "Discriminant Analysis Tutorial," *http://people.revoledu.com/kardi/tutorial/LDA/index.html*.
[5] V. Vapnik, *The Nature of Statistical Learning Theory*. Springer, 1995.
[6] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo, "A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabeled Data," in *Applications of Data Mining in Computer Security*, 2002.
[7] Z. Liu and I. Elhanany, "A QoS-Aware Reinforcement Learning based MAC Protocol for Wireless Sensor Networks," in *Proc. of Intl. Conference on Networking, Sensing, and Control*, April 2006.
[8] K. Steinhaeuser, N. V. Chawla, and C. Poellabauer, "Towards Learning-based Sensor Management," in *Proc. of Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SYSML), SIGMETRICS*, June 2004.
[9] Y. Zhang and Q. Huang, "A Learning-based Adaptive Routing Tree for Wireless Sensor Networks," *Journal of Communications*, vol. 1, no. 2, May 2006.
[10] C. E. Loo, M. Y. Ng, C. Leckie, and M. Palaniswami, "An Intrusion Detection for Routing Attacks in Sensor Networks," *Intl. Journal of Distributed Sensor Networks*, 2005.
[11] R. Duda, P. Hart, and D. Stork, *Pattern Classification*. Wiley-Interscience, 2001.
[12] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer, 2001.
[13] A. Silva, M. Martins, B. Rocha, A. Loureiro, L. Ruiz, and H. Wong, "Decentralized Intrusion Detection in Wireless Sensor Networks," in *Proc. of Intl. Workshop on Quality of Service and Security in Wireless and Mobile Networks*, October 2005, pp. 16–23.
[14] K. Ioannis and T. Dimitriou, "Towards Intrusion Detection in Wireless Sensor Networks," in *Proc. of European Wireless Conference*, 2007.
[15] W. Xu, W. Trappe, Y. Zhang, and T. Wood, "The Feasibility of Launching and Detecting Jamming Attacks in Wireless Networks," in *Proc. of MobiHoc*, 2005.