An Approach for Detecting and Distinguishing Errors versus Attacks in Sensor Networks

Claudio Basile[*], Meeta Gupta[†], Zbigniew Kalbarczyk, Ravi K. Iyer
Center for Reliable and High-Performance Computing
University of Illinois at Urbana-Champaign, IL 61801
{basilecl,mgupta2,kalbar,iyer}@crhc.uiuc.edu

Abstract

Distributed sensor networks are highly prone to accidental errors and malicious activities, owing to their limited resources and tight interaction with the environment. Yet only a few studies have analyzed and coped with the effects of corrupted sensor data. This paper contributes with the proposal of an on-the-fly statistical technique that can detect and distinguish faulty data from malicious data in a distributed sensor network. Detecting faults and attacks is essential to ensure the correct semantic of the network, while distinguishing faults from attacks is necessary to initiate a correct recovery action. The approach uses Hidden Markov Models (HMMs) to capture the error/attack-free dynamics of the environment and the dynamics of error/attack data. It then performs a structural analysis of these HMMs to determine the type of error/attack affecting sensor observations. The methodology is demonstrated with real data traces collected over one month of observation from motes deployed on the Great Duck Island.

1. Introduction

Distributed sensor networks are being deployed for a wide range of applications, such as surveillance, habitat monitoring, and health care. Owing to their limited resources and tight interaction with the environment, sensor nodes can report corrupt readings due to environmental disturbances, accidental faults in the sensor hardware or software, or malicious activities, such as an adversary capturing and reprogramming a number of sensor nodes. Field studies indicate that a major cause of failure are the errors originating in degraded sensor devices, which directly interact with the environment and are thus subjected to a variety of physical, chemical, and biological forces. Interestingly, these sensor errors are likely to manifest days before the sensor electronics actually fail [1].

The need for reliable and secure operation in critical sensor applications is compelling. While most research focuses on protecting a sensor network from network-level errors and attacks (e.g., denial-of-service [2], malicious message routing and forwarding [3]), few studies have analyzed and proposed solutions to cope with the effects of corrupted/malicious sensor data [4]. This paper contributes with an on-the-fly statistical technique that not only can detect anomalous sensor data but also can distinguish faulty data from malicious data in a distributed sensor network. Detecting faults and attacks is essential to ensure the correct semantic of a network, while distinguishing faults from attacks is necessary to diagnose the detected anomaly and initiate a correct recovery action.

The proposed approach employs Hidden Markov Models (HMMs). HMMs have been widely used in the areas of speech recognition, gesture recognition, and gene finding. More recently, HHMs have been applied in the domain of host-based intrusion detection [5,6]. In this work, we propose using HMMs for anomaly detection, but we take an entirely new direction. First, we target distributed networked systems, sensor networks in the current paper, and we leverage the intrinsic redundancy that these systems provide to overcome the complexity of the classical HMM identification problem. At each time step of our algorithm, multiple, correlated observations are available from multiple sources. Assuming that the errors or the adversary have not compromised (yet) a majority of the sources, we can use a statistical clustering-based technique to (statistically) separate correct observations from faulty/malicious observations. Thus, we can efficiently identify the HMM $\mathcal{M}_{CO}$ capturing the correspondence between the hidden and correct dynamics of the sensed phenomenon P (derived using only data from correct sources) and the observable dynamics of P (derived using all collected data). Importantly, we do not require a separate training phase in which the system is left vulnerable to incoming errors and attacks, because at no point in time do we require all incoming data to be error/attack-free. Second, while prior work has used HMMs merely to detect anomalies, we use these powerful mathematical models also to diagnose and characterize the nature of the detected anomalies, i.e., to distinguish errors versus attacks and to determine the type of error or the type of attack that affects the system. We accomplish this goal by tracking the error/attack behavior with an additional HMM $\mathcal{M}_{CE}$ that captures the correspondence between the hidden dynamics of P and the dynamics of error/attack data, and by performing a structural analysis of the two HMMs. We demonstrate our methodology with an experimental study that uses real data

[*]C. Basile is currently with Bell Laboratories, Holmdel, NJ 07733, and can be reached at cbasile@lucent.com.

[†]M. Gupta is currently with the Division of Engineering and Applied Sciences at Harvard University, Cambridge, MA 02138, and can be reached at meeta@eecs.harvard.edu.

traces collected over one month of observation from motes deployed on the Great Duck Island [7].

2. Overview of Hidden Markov Models

A Hidden Markov Model (HMM) [8] captures a hidden stochastic process that is inferred through a sequence of observations, which are stochastically related to the state of the hidden process. Mathematically, an HMM is characterized by: (1) a set of hidden states $S_1, \ldots, S_M$; (2) a set of observation symbols $V_1, \ldots, V_N$; (3) a state transition probability distribution $\mathbf{A} = \{a_{ij}\}_{ij}$, where $a_{ij} = Pr\{s_{t+1} = S_j | s_t = S_i\}$ and s_t is the hidden state at time t; (4) an observation symbol probability distribution $\mathbf{B} = \{b_{ik}\}_{ik}$, where $b_{ik} = Pr\{v_t = V_k | s_t = S_i\}$ and v_t is the observation symbol at time t; and (5) an initial state distribution $\pi = \{\pi_i\}_i$, where $\pi_i = Pr\{s_o = S_i\}$ and s_o is the initial hidden state.

HMMs have recently been used to detect malicious intrusions in a single-host computer system by tracing the system calls invoked by a monitored application run on the system [5,6]. In an initial attack-free learning phase, an HMM λ is identified to capture the correct behavior of the application. The approach in [5] assumes a hidden state for each system call (around 40 in the reported experiments), which leads to a large training time (about 2 weeks) due to the high complexity of the standard HMM identification problem. In a subsequent testing phase, the probability $Pr\{O|\lambda\}$ that the observed application behavior O is produced by model λ is computed, and an anomaly is detected if this probability is less than a given threshold η. This simple method suffers from a series of limitations: (1) The choice of the hidden states of the HMM is arbitrary, difficult to justify, and brings about states that have no physical interpretation. (2) The required assumption of an attack-free training phase and long training time are unpractical for real-world applications, where the intrusion detection system must be periodically re-tuned to account for natural variations in the monitored system's behavior. (3) The approach is not designed for distributed environments. By exploiting the redundancy naturally present in sensor networks, this paper uses HMMs in an entirely new way and, thus, overcomes the limitations mentioned above.

3. Proposed Methodology

This section discusses the proposed methodology for detecting, diagnosing and classifying errors and attacks in a distributed network. Before introducing the technical details of the methodology, we outline its basic steps. Data streams from sensors deployed in a region of interest R are used as the input to the analysis procedure (see Fig. 1). The proposed procedure executes on a single data collector node (e.g., a base station or a cluster head) and operates as follows:

1. Collect sensor observations and group them according to a predefined time window w.
2. From the observations collected in each time window and a set of potential states of the environment (obtained as discussed below) generate: (i) a sequence o_i of the observable states of the sensed environment (derived using all the collected data regardless of their correctness), (ii) a sequence c_i of the hidden states of the environment (i.e., actual, unknown states traversed by the environment, e.g., error-free temperature values), and (iii) a sequence e_i of the erroneous states traversed by the observations that are (potentially) corrupted by accidental errors or malicious attacks.
3. Build an HMM $\mathcal{M}_{CO}$ that relates the hidden states of the environment (indicated by c_i) with observable states of the environment (indicated by o_i), and an HMM $\mathcal{M}_{CE}$ that relates the hidden states of the environment with the error/attack states (indicated by e_i).
4. Analyze the two HMMs and, based on a library of known error/attack models, classify the type of error/attack that has affected the original observations.
5. Extract a Markov Model $\mathcal{M}_C$ to provide the user with an error/attack-free description of the dynamics of the environment.

3.1. Data Collection and Preprocessing

Figure 1 depicts the key steps (represented as rectangles in the figure) of the proposed methodology. We assume that sensors are multimodal, i.e., they can measure different types of physical attributes, such as temperature, pressure, and humidity. We model the value of the environment attributes monitored by sensors in the region of interest R as a multidimensional, unknown parameter $\Theta(t)$ that changes with time. We assume that each sensor periodically sends a message $\langle t, p \rangle$ to a single collector node, where t is the time in which the readings are taken and $p = \langle x_1, \ldots, x_n \rangle$ is the vector of n environment attributes sampled at time t. We assume that the signal p_j sensed by a correct sensor j can be modeled as $p_j = \Theta(t) + \mathcal{N}_j$, where $\mathcal{N}_j$ is a zero-mean, random measurement noise.[1] Note that in contrast with classical estimation theory, we let a number of these sensor observations be arbitrarily corrupted, owing to faults/attacks.

Given a set of collected sensor observations $\mathcal{O} = \{\langle t, p \rangle\}$, the collector node partitions these observations into time intervals of duration w so as to build a sequence of observation sets $\{\mathcal{O}_i\}$ such that:

$$\mathcal{O}_i = \{p | \langle t, p \rangle \in \mathcal{O} \wedge w \cdot (i-1) \leq t \leq w \cdot i\}. \quad (1)$$

Parameter w must be large enough to create nonempty sets $\mathcal{O}_i$ yet small enough to enable us to accurately sample changes in the environment attributes $\Theta(t)$ (e.g., $\Theta(t)$ should be approximately constant in an observation window w).

[1]More complex environment estimation problems are the object of future extensions.

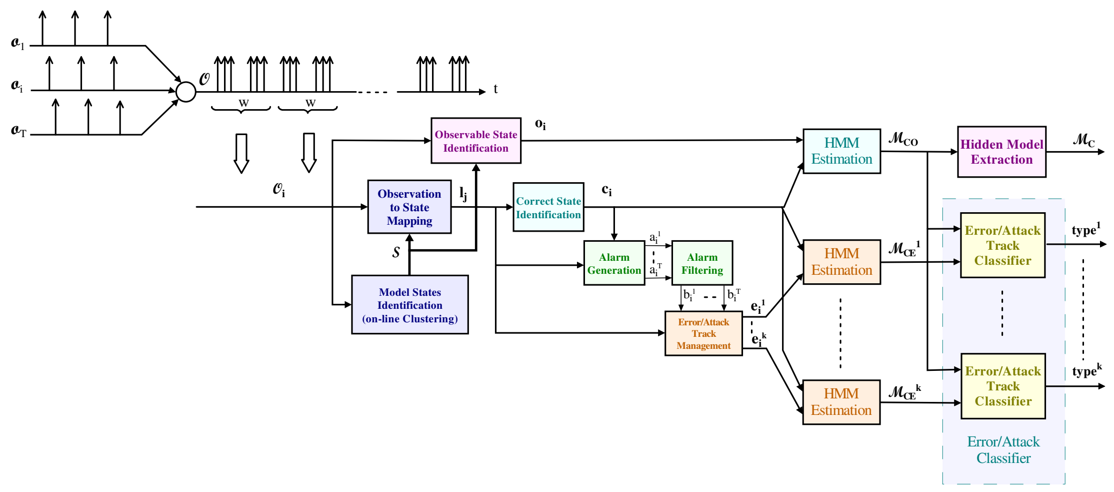


Figure 1. Methodology for error/attack detection and classification.

A *Model State Identification* module uses an on-line statistical clustering algorithm to identify the possible states $\mathcal{S} = \{s_1, \ldots, s_M\}$ of the environment, which we use to synthetically describe the physical conditions traversed by the sensed phenomenon and by error/attack data. Figure 2 depicts an execution scenario where eight states s_j are identified. For clarity of presentation, the operation of the Model State Identification module is discussed at the end of this section.

An *Observable State Identification* module uses a current observation set $\mathcal{O}_i$ to determine the current observable state of the environment o_i, i.e., the state that best describes the totality of the observations p_1,...,p_N in $\mathcal{O}_i$:

$$o_i = \arg \min_{1 \le k \le M} \|s_k - \frac{1}{N}\sum_{j=1}^{N} p_j\|. \tag{2}$$

Figure 3 depicts an execution scenario where an observation set $\mathcal{O}_i$ of six observations p_j is mapped to the observable state s_2 (hence, $o_i = 2$), since s_2 is the closest state to the mean value calculated across all observations.

An *Observation to State Mapping* module maps each observation p_j in $\mathcal{O}_i$ to the model state that best represents p_j. Formally, the module computes:

$$l_j = \arg \min_{1 \le k \le M} \|s_k - p_j\|. \tag{3}$$

In Fig. 4, observations p_1 to p_4 are closest to state s_1; thus, $l_1 = l_2 = l_3 = l_4 = 1$. On the other hand, observations p_5 and p_6 are closest to states s_4 and s_5, respectively; hence, $l_5 = 4$ and $l_6 = 5$.

A *Correct State Identification* module determines the correct environment state c_i, i.e., the state that best describes the largest group of observations in $\mathcal{O}_i$ that cluster together:

$$c_i = \arg \max_{1 \le k \le N} |\{p_j \in \mathcal{O}_i | l_j = k\}| \tag{4}$$

In the example of Fig. 4, the correct state is s_1 (hence, $c = 1$), since it is associated with the largest subset of observations p_j. To guarantee valid operation of this module, it is assumed that the largest set of observations that cluster together always includes a majority of correct observations. Intuitively, this corresponds to saying that correct observations both behave alike (i.e., are not split into a number of small-size clusters) and exceed faulty/malicious observations. While it is possible to craft pathological scenarios in which the assumption is violated, the remainder of this paper assumes that the system parameters are properly tuned (e.g., the Model State Identification module does not generate too many model states) and that a majority of sensors have not been compromised (yet) by errors and attacks.

An *Alarm Generation* module checks whether a sensor j belongs to the correct state c and generates a (raw) alarm a^j if the reading from that sensor does not belong to the correct state. In the example of Fig. 4, two alarms are generated for sensors 5 and 6.

An *Alarm Filtering* module filters the generated alarms to reduce the false-alarm probability. A simple approach is to generate a filtered alarm only after receiving k raw alarms in the last n time steps (with $k \le n$). Sophisticated approaches can leverage change detection schemes such as Sequential Probability Ratio Test (SPRT) and Cumulative Sum (CUMSUM) procedure [9].

Filtered alarms are passed to an *Error/Attack Track Management* module. Since different sensors may be affected by different types of errors/attacks, we maintain a separate error/attack track e for each of the sensors. In the proposed mechanism, if a filtered alarm b^j is raised and there is no currently active error/attack track for sensor j, then a new track e^{t+1} is open for that sensor (where t is the number of tracks that were previously active). If a filtered alarm b^j is cleared, then the error/attack track associated with the sensor j is closed. For each sensor j for which there is an active error/attack track e^k, the Error/Attack Track Management module sets $e_i^k = l_k$ if at time i sensor value p_k is not mapped

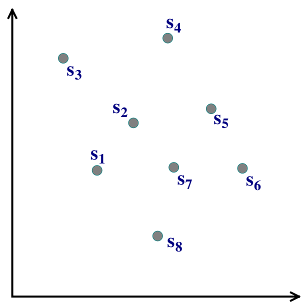


Figure 2. Identification of the model states.

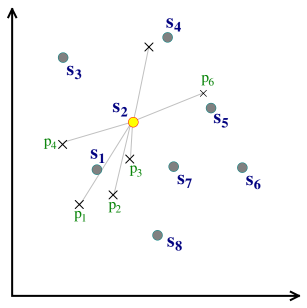


Figure 3. Identification of the observable environment state.

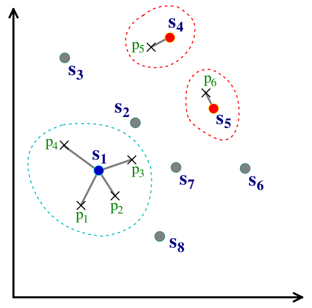


Figure 4. Identification of the correct environment state and the error/attack states.

to the correct state (i.e., $l_k \neq c_i$), or $e_i^k = \perp$ otherwise. State $\perp$ is a fictitious state that is used to model the cases in which a sensor for which there is an open error/attack track generates data in agreement with correct sensors. In the example of Fig. 4, sensor 5 and sensor 6 are mapped to the error/attack states s_4 and s_5, respectively.

The procedure presented above uses a set of model states $\mathcal{S} = \{s_1, \ldots, s_M\}$ to synthetically describe the possible physical conditions traversed by the environment and by the error/attack data. The goal of the *Model State Identification* module is to provide an updated estimate of this set. To capture natural variations in the model states, the Model State Identification module uses an on-line statistical clustering algorithm. Starting from an initial set of model states S_o (e.g., selected randomly or based on historical data), the module uses the incoming observation set $\mathcal{O}_i$ to update the value of these states. The goal is to obtain a small set of states that best represents the (potentially time-varying) distribution of the incoming data. For each state s_k, the module first computes the set of observations that are mapped to s_k

$$P_k = \{p_j \in \mathcal{O}_i | l_j = k\} \tag{5}$$

and then, if P_k is not empty, state s_k is updated as follows

$$s_k = (1-\alpha)s_k + \frac{\alpha}{|P_k|} \sum_{p_j \in P_k} p_j \tag{6}$$

where α is a learning factor ranging in the open interval $(0, 1)$. In the example of Fig. 4, $P_1 = \{p_1, p_2, p_3, p_4\}$, $P_4 = \{p_5\}$, and $P_5 = \{p_6\}$.

From the discussion above, it emerges that the Model State Identification module should not split correct data into a number of small-size clusters. This can be achieved by merging two states that are too close to each other (relative to a predefined threshold) into a single state. Similarly, the module should expand the current set of states when appropriate. This can be achieved by checking if an observation p_j is too far (relative to a predetermined threshold) from its corresponding state s_{l_j}[2] and by creating a new state $s_{M+1} = p_j$ accordingly.

[2] l_j is the state corresponding to p_j, as calculated by using Equation 3.

3.2. Environment Modeling through HMMs

The proposed methodology analyzes the structural properties of two HMMs (see § 2 for an overview of HMMs) estimated from the collected data to classify the nature of the error or attack that affects the system. In a general HMM, hidden states and observation symbols may be of a different nature. In the methodology discussed in § 3, however, both hidden states and observation symbols reflect the possible physical states $\{s_1, \ldots, s_M\}$ of the sensed environment, which are estimated by the Model States Identification module in Fig. 1. Also, the proposed methodology considers two types of HMMs: (1) an HMM $\mathcal{M}_{CO}$ that relates sequence c_i (representing the hidden/correct dynamics of the environment) with sequence o_i (representing the observable dynamics of the environment); and (2) an HMM $\mathcal{M}_{CE^k}$ that relates sequence c_i with an error/attack track e_i^k (representing the dynamics of an erroneous/malicious sensor). Note that hidden/correct states of the environment c_i are not directly observable. In the proposed methodology, they are estimated by a Correct State Identification module and are, thus, available when building the two types of HMMs.

In this context, we can use a simple on-line procedure to estimate an HMM.[3] The proposed procedure operates at the end of each observation time window and uses the estimate of the current hidden state of the model (c_i) and the current observation symbol (either o_i or e_i^k, depending on the considered HMM). Let j be the current state, i be the state at the previous time step, and l be the current observation symbol. Then, at each time t we execute the following steps:

- If $j \neq i$, update the state transition distribution: $\forall k = 1..M : a_{ik} = (1-\beta)a_{ik} + \beta\delta_{kj}$.
- Update the observation symbol distribution: $\forall k = 1..M : b_{ik} = (1-\gamma)b_{ik} + \gamma\delta_{kl}$.

In the above equations, β and γ are learning factors ranging in the open interval (0,1), and δ_{ij} is Kronecker's delta.[4] At initialization time, matrices $\mathbf{A}$ and $\mathbf{B}$ can be set equal to identity

[3] Advanced on-line HMM estimation techniques can be found in [10].
[4] δ_{ij} is 1 if i is equal to j, and 0 otherwise.

matrices. It is easy to show that if $\mathbf{A}$ and $\mathbf{B}$ are probability distributions, then they remain so when updated with the two formulas above.

3.3. Error and Attack Models

Sensor nodes can report corrupt data due to environmental disturbances, accidental errors in the sensor hardware or software, or malicious attacks, such as an adversary capturing and reprogramming a number of sensor nodes. In order to initiate a proper recovery action we need to differentiate between random errors and malicious tampering with the system. Strictly speaking, distinguishing errors from attacks is not always possible, since an adversary can launch an attack that behaves like an error. In this work, we make an attempt to answer the following question: "Given a detected network malfunctioning, can we determine the type of the underlying condition, accidental or malicious, that most likely caused this misbehavior?" The proposed approach exploits the notion and the properties of the HMMs introduced in § 3.2. Before presenting the theoretical and technical details of our error/attack classification methodology, we introduce a formal description of errors and attacks in sensor networks.

Model for Accidental Errors in Sensor Networks. Sensor devices interact directly with the environment and, hence, are subjected to a variety of physical, chemical, and biological forces; this can degrade them fairly quickly. Field studies [1] indicate that errors originating in degraded sensor devices are a major cause of unreliability in a wireless sensor network. Interestingly, these sensor failures are likely to manifest days before the sensor electronics may fail. Based on these results, we assume the following *sensor fault model*:

- *Stuck-at-Value Error*, a faulty sensor i constantly reports a fixed reading;
- *Calibration Error*, a faulty sensor i's readings are affected by a multiplicative error;
- *Additive Error*, a faulty sensor i's readings are affected by an additive error;
- *Random Noise Error*, a faulty sensor i's readings are affected by a zero-mean noise with high variance.

It is possible that the network is affected by an error whose nature is different from the models above. In that case, we say that the network is affected by an *Unknown Error*.

Model for Malicious Attacks in Sensor Networks. An adversary may be interested in disrupting the communication infrastructure of the network (e.g., by jamming wireless transmission) or in disrupting/controlling the environmental sensing mechanism of the network (e.g., by tampering with existing sensors to inject malicious data in the network). In this work, we focus on the latter type of attack because it is not easily detectable (e.g., by monitoring the arrival rate of sensor data) and because it directly affects the semantics of the sensor network. Based on these observations, we assume the following *sensor attack model*:

- *Dynamic Creation*, an adversary is introducing a spurious behavior in the sensed environment, e.g., while correct sensors report constant temperature readings, the adversary injects high temperature values so that the overall temperature measured by the network moves from the valid readings and increases substantially;
- *Dynamic Deletion*, an adversary is removing a valid behavior in the sensed environment, e.g., when correct sensors start to report an increase in the environment temperature, the adversary injects low temperature values so that the overall temperature measured by the network does not change;
- *Dynamic Change*, an adversary is modifying the attributes of an environment physical state, e.g., the adversary selectively injects low temperature values so that each time correct sensors report a 50 value in the environment temperature the overall temperature measured by the network equals 10;
- *Mixed*, an adversary is mounting a combination of the attacks described above.

Distinguishing a simple attack (i.e., a Creation, a Deletion, or a Change attack) from a more complex one (i.e., a Mixed attack) can reveal valuable insights about the adversary. For instance, to pass undetected, an adversary may be interested in deleting environment changes that would otherwise be generated by a malicious activity that the adversary is inducing in the environment. Precise knowledge of those changes can help reveal the nature of the mounted attack.

The aim of our study is to classify attacks that change the observable behavior of a system; hence, our attack models focus on such attacks. A *benign* attack where the attacker behaves according to correct sensors' behavior is not a type of attack we classify using our methodology, for it does not alter the system behavior in any manner. Similarly, different attacks at the network layer, like packet dropping, denial of service, etc., are not addressed by our approach. Different techniques have been developed to tackle network-level attacks, and the discussion of those techniques is beyond the scope of our paper.

3.4. Error versus Attack Classification

The proposed error/attack classification methodology is portrayed in Fig. 5. To diagnose and classify system Malfunctioning, we construct two mathematical models that capture the system's dynamic behavior, namely an HMM $\mathcal{M}_{CO}$ relating correct environment states to observable environment states, and an HMM $\mathcal{M}_{CE}$ relating correct environment states to error/attack states. Occurrence of errors and attacks in the sensor network alter the dynamics of the sensed environment.

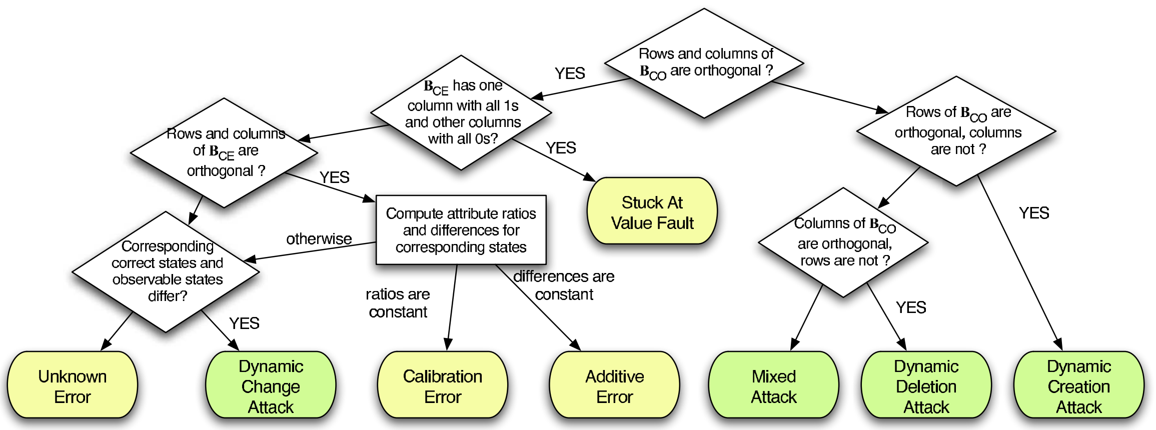

Rows and columns of $\mathbf{B}_{CO}$ are orthogonal ?
YES
$\mathbf{B}_{CE}$ has one column with all 1s and other columns with all 0s?
YES
Stuck At Value Fault
Rows and columns of $\mathbf{B}_{CE}$ are orthogonal ?
YES
Compute attribute ratios and differences for corresponding states
otherwise
Corresponding correct states and observable states differ?
YES
ratios are constant
differences are constant
Rows of $\mathbf{B}_{CO}$ are orthogonal, columns are not ?
Columns of $\mathbf{B}_{CO}$ are orthogonal, rows are not ?
YES
YES
Unknown Error
Dynamic Change Attack
Calibration Error
Additive Error
Mixed Attack
Dynamic Deletion Attack
Dynamic Creation Attack

Figure 5. Error/Attack classification methodology.

These anomalies are reflected in anomalies in the two HMMs, and hence, can be detected by inspecting these models.

To distinguish faults from attacks, we want to exploit the following intuition: Attacks are generated by an intelligent entity that *knows* the underlying dynamics of the environment and attempts to *selectively* change the view of the environment that is sensed by the network. We express this intuition by assuming, as a first-order approximation, that "attacks change the temporal behavior of the environment as sensed by the network, while errors do not." (Only Dynamic Change attacks are not covered by this assumption and require additional distinction.) From our perspective, making this assumption corresponds to saying that if we build a Markov Model $\mathcal{M}_C$ from the sequence of the correct environment states c_i and a Markov Model $\mathcal{M}_O$ from the sequence of the observable environment states o_i, then in case of errors the two models have the same number of states and the same set of transitions, while they may have different attributes (e.g., temperature, humidity) associated with a given state. In the proposed approach, we do not build and compare such two Markov Models, but more conveniently, we check whether the rows and the columns of the observation symbol probability distribution $\mathbf{B}^{CO}$ of the HMM $\mathcal{M}_{CO}$ are orthogonal: $\forall i,j : \sum_k b^{co}_{ik} b^{co}_{jk} = \delta_{ij}$ and $\forall i,j : \sum_k b^{co}_{ki} b^{co}_{kj} = \delta_{ij}$. The first equation expresses the condition that if two hidden states are different, then they generate two different observation symbols. The second equation expresses the condition that if two observation symbols are different, then they are generated by two different hidden states. Recall that in our methodology we define hidden states of $\mathcal{M}_{CO}$ as correct environment states, and observation symbols of $\mathcal{M}_{CO}$ as observable environment states (see § 3.2). Further classification into attack types and error types requires a more in-depth look at the HMM $\mathcal{M}_{CO}$ for attacks and at the HMM $\mathcal{M}_{CE}$ for errors.

Attack Type Determination. To determine the type of a detected attack, we look at the HMM $\mathcal{M}_{CO}$. Through this model, we can study the effect of the attack on the observable state of the environment as a function of the correct states of the environment. In the absence of errors and attacks, each correct state of the environment (hidden state of $\mathcal{M}_{CO}$) corresponds to a single observable state of the environment (observation symbol in $\mathcal{M}_{CO}$). The presence of an attack results in a change of this one-to-one correspondence.

A Dynamic Creation attack is characterized by a correct environment state being associated with multiple observable environment states (say, states i and j). Formally, this corresponds to saying that there are two columns i and j of $\mathbf{B}^{CO}$ that are not orthogonal:$\exists i,j : \sum_k b^{co}_{ki} b^{co}_{kj} \neq 0$. A Dynamic Deletion attack is characterized by multiple correct environment states (say, states i and j) being associated with the same observable environment state. Formally, this corresponds to saying that there are two rows i and j of $\mathbf{B}^{CO}$ that are not orthogonal: $\exists i,j : \sum_k b^{co}_{ik} b^{co}_{jk} \neq 0$. If the conditions above hold at the same time, then we classify a detected attack as a Mixed attack.

A Dynamic Change attack is characterized by a correct environment state being associated with a single observable environment state, and vice versa. The attack does not affect the orthogonality of rows and columns of $\mathbf{B}^{CO}$, and thus, the classification of this attack is shown on the left-hand side of Fig. 5. To classify a Dynamic Change attack, we need to look at the values of the attributes of corresponding correct and observable states of $\mathcal{M}_{CO}$. In the presence of such an attack, a correct state $s^c = \langle x^c_1, ..., x^c_n \rangle$ associated with an observable state $s^o = \langle x^o_1, ..., x^o_n \rangle$ in $\mathcal{M}_{CO}$ is such that $\forall i : x^c_i \neq x^o_i$, which reflects the attackers intention of modifying the physical attributes of the sensed environment without changing its temporal behavior.

Error Type Determination. To determine the type of a detected error, we look at the HMM $\mathcal{M}_{CE}$. Through this model, we can study the behavior of the error as a function of the correct state of the environment. Recall that in our methodology we define hidden states of $\mathcal{M}_{CE}$ as correct environment states, and observation symbols of $\mathcal{M}_{CE}$ as error/attack states (see § 3.2).

A Stuck-at-Value error is characterized by a faulty sensor always reporting the same value, independently of the correct state of the environment. Thus, all correct environment states are associated with the same error state. Formally, this corresponds to saying that the observation symbol probability distribution $\mathbf{B}^{CE}$ of $\mathcal{M}_{CE}$ is such that it has one column (k) that has all ones and other columns of all zeros:

$$\exists k : \forall i : b_{ij}^{ce} = \begin{cases} 1 & \text{if} \quad j = k \\ 0 & \text{if} \quad j \neq k \end{cases} . \tag{7}$$

A Calibration error is characterized by a faulty sensor reporting an erroneous value that changes accordingly with the correct state of the environment. Thus, there is a one-to-one mapping between correct states and error states. This also holds true for an Additive error. Formally, this corresponds to saying that the rows and the columns of the observation symbol probability distribution $\mathbf{B}^{CE}$ are orthogonal:

$$\forall i, j : \sum_k b_{ik}^{ce} b_{jk}^{ce} = \sum_k b_{ki}^{ce} b_{kj}^{ce} = \delta_{ij}. \tag{8}$$

To further classify between additive and calibration errors, we need to compute the ratio and the difference between the attributes of corresponding correct and error/attack states in $\mathcal{M}_{CE}$. A Calibration error leads to a constant ratio, while an Additive error leads to a constant difference. Formally, given a correct state $s^c = \langle x_1^c, ..., x_n^c \rangle$ associated with an error/attack state $s^e = \langle x_1^e, ..., x_n^e \rangle$ in $\mathcal{M}_{CE}$, there is a constant $K = \langle k_1, ..., k_n \rangle$ such that $\forall i : \frac{x_i^c}{x_i^e} = k_i$ for a Calibration error or $\forall i : x_i^c - x_i^e = k_i$ for an Additive error. If neither of the conditions holds, then we check for the presence of a Dynamic Change attack (described above).

Under the environment estimation problem considered in this paper ($p_j = \Theta(t) + \mathcal{N}_j$, see § 3.1), it is difficult to correctly classify a Random Noise error, as there is no fixed pattern observed in the observation symbol probability distribution, and the $\mathcal{M}_O$ and $\mathcal{M}_C$ estimated are identical. Thus, a random error can be misclassified as being in an error-free system state.

4 Experimental Results and Discussion

The next sections present results from an experimental study where the proposed methodology is applied to real data traces collected over one month of observation from 10 motes (sensor nodes) deployed on the Great Duck Island (GDI) [7]. The GDI testbed consists of around 32 motes, out of which 9 are burrow motes. Our experiments only use the data gathered by the outside motes. All motes are capable of measuring environment attributes, such as temperature, humidity, and pressure, and all sample the environment at 5 minute intervals. Due to the presence of missing and malformed sensor packets, when applying our methodology the constructed models may present spurious states. In this initial evaluation, we do not eliminate such states.

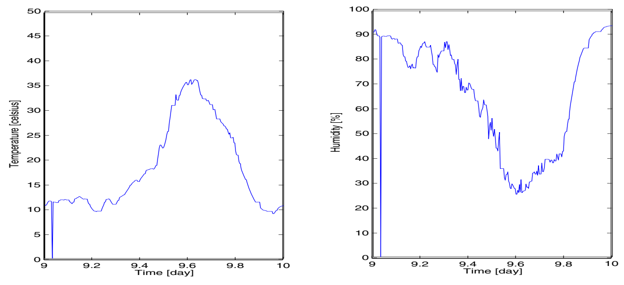


Figure 6. Humidity and temperature variation for July 9th.

Parameter	Description	Value
K	Number of sensors	10
M	Number of initial model states	6
w	Observation window size	12
α	Learning factor used to estimate model states	0.10
β	Learning factor used to estimate state transition probability $\mathbf{A}$	0.90
γ	Learning factor used to estimate observation symbol probability $\mathbf{B}$	0.90

Table 1. Parameters used in the experimental setup.

4.1 Fault Classification

This section analyzes the GDI data for the whole month of July 2003. Figure 6 shows the variation of temperature and humidity respectively for one complete day, July 9th. We can clearly observe that the temperature and humidity change continuously during the day. A similar trend is observed for the whole month.

Table 1 lists the experimental setup used in determining the Markov Models using the procedure discussed in § 3.1. The Model State Identification module requires an initial estimate for the set of model states. This initial estimate can be completely random or based on historical data. The results discussed in the paper are based on an initial set estimate of 6 states that is determined by running an off-line clustering algorithm on the entire data.[5] As mentioned earlier, the observation window size is also an important input to the system. Since the sensors in our setup sample data every 5 minutes, we use a window size of 12 samples, which enables us to capture variations in the environment attributes with sufficient time accuracy (one hour accuracy) and statistical significance (about a hundred sensor readings in average, as not all sensor data can be used due to missed or corrupted packets).

Figure 7 depicts the correct Markov Model $\mathcal{M}_C$ of the environment, as estimated by the procedure delineated in § 3 (see Fig. 1). Four main states of the system can be identified, namely (12,94), (17,84), (24,70) and (31,56), where each state is represented by a (temperature value, humidity value) tuple. An additional state (16,27) results from fluctuations within the

[5]The proposed methodology worked equally well when a set of random initial states was provided to the Model State Identification module.

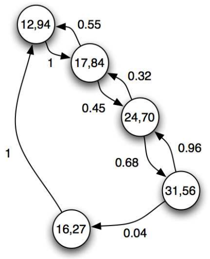


Figure 7. Correct Markov Model $\mathcal{M}_C$ of the environment.

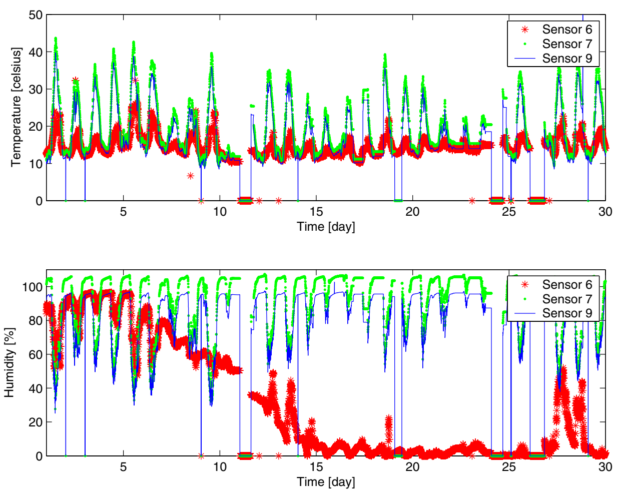
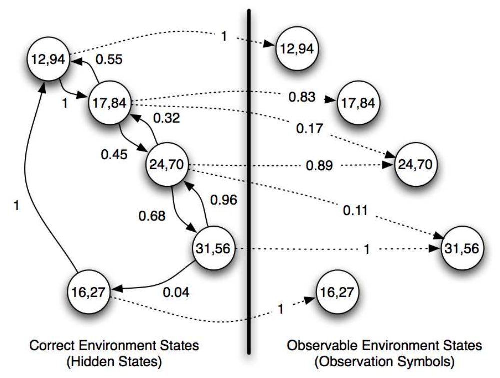
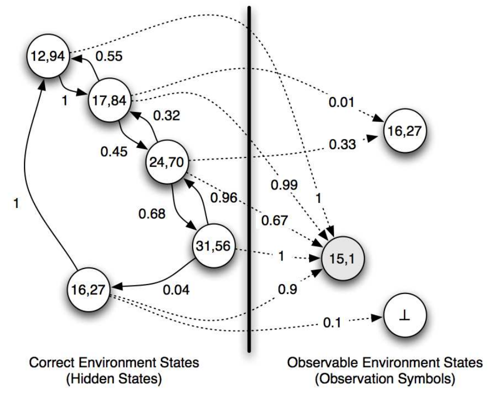


Figure 8. Humidity and temperature variation in a week for sensors 6, 7, and 9.



Correct Environment States (Hidden States)
Observable Environment States (Observation Symbols)

(a) $\mathcal{M}_{CO}$.



Correct Environment States (Hidden States)
Observable Environment States (Observation Symbols)

(b) $\mathcal{M}_{CE}$.

Figure 9. HMMs for faulty sensor 6 (stuck-at-value fault).

readings. The transition to this state has a very low probability, and hence, this state is not further considered as one of the key states of the system.

By applying the proposed methodology to the GDI data, we discover two sensor nodes to be consistently faulty, namely sensor 6 and sensor 7. Figure 8 depicts the humidity values reported by the two sensors as compared with a non-faulty sensor 9. As seen in the figure, sensor 6 starts reporting a continuously decreasing value of the humidity that eventually leads in an almost-zero value, whereas sensor 7 reports, on average, a value about 10% higher than the correct sensors.

Figure 9 depicts the two Hidden Markov Models $\mathcal{M}_{CO}$ and $\mathcal{M}_{CE}$ learned for sensor 6. Table 2 and Table 3 report the corresponding state transition probability matrix ($\mathbf{A}$) and observation symbol probability matrix ($\mathbf{B}^{CO}$ and $\mathbf{B}^{CE}$). The additional state ($\perp$) introduced in $\mathbf{B}^{CE}$ (see Table 3) models the scenario when the faulty sensor does not produce faulty data. This fictitious state is not taken into account during classification. Based on the relations described in § 3.4, we observe that the rows and the columns of $\mathbf{B}^{CO}$ are approximately orthogonal ($\sum_k b^{co}_{ik} b^{co}_{jk} < 0.1$ for $i \neq j$, and $\sum_k b^{co}_{ik} b^{co}_{jk} > 0.8$ for $i = j$). Also, Table 3 shows that matrix $\mathbf{B}^{CE}$ has all columns approximately null apart from a single column (corresponding to state (15,1)) of approximately all ones. This leads to (correctly) classifying sensor 6 to be in a stuck-at state (15,1).

$i \downarrow, j \rightarrow$	(12,94)	(31,56)	(16,27)	(24,70)	(17,84)
(12,94)	1	0	0	0	0
(31,56)	0	1	0	0	0
(16,27)	0	0	1	0	0
(24,70)	0	0.11	0	0.89	0
(17,84)	0	0	0	0.17	0.83

Table 2. $\mathbf{B}^{CO}$ matrix for faulty sensor 6 (stuck-at-value fault).

$i \downarrow, j \rightarrow$	(16,27)	(15,1)	$\perp$
(12,94)	0	1	0
(31,56)	0	1	0
(16,27)	0	0.9	0.1
(24,70)	0.33	0.67	0
(17,84)	0.01	0.99	0

Table 3. $\mathbf{B}^{CE}$ matrix for faulty sensor 6 (stuck-at-value fault).

A similar analysis can be conducted for sensor 7, where Table 4 and Table 5 show the resulting observation symbol probability matrices, $\mathbf{B}^{CO}$ and $\mathbf{B}^{CE}$. Both matrices are ap-

$i \downarrow, j \rightarrow$	(22,80)	(17,88)	(13,96)	(27,68)	(32,56)
(22,80)	0.98	0.02	0	0	0
(17,88)	0	0.8	0.2	0	0
(13,96)	0	0	1	0	0
(27,68)	0.001	0	0	0.999	0
(32,56)	0	0	0	0.001	0.999

Table 4. B^{CO} matrix for faulty sensor 7 (calibration fault).

$i \downarrow j \rightarrow$	(22,80)	(17,88)	(13,96)	(27,68)	(32,56)	$\perp$
(22,80)	0	0.86	0	0	0	0.14
(17,88)	0	0	0.85	0	0	0.15
(13,96)	0	0	0	0	0	1
(27,68)	0.87	0	0	0	0	0.13
(32,56)	0	0	0	0.46	0	0.54

Table 5. B^{CE} matrix for faulty sensor 7 (calibration fault).

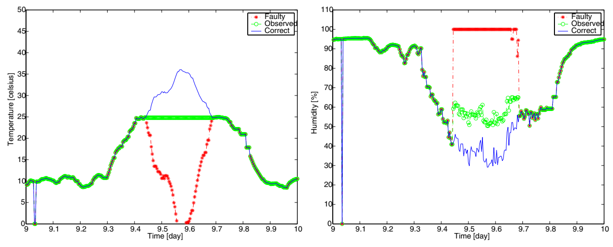
proximately orthogonal. Furthermore, when computing the ratios x_i^c/x_i^e and the differences $x_i^c - x_i^e$ between the attributes of corresponding model states,[6] we obtain ratios with average (1.24, 1.16) and low variance (0.006,0.007), and differences with average (5,10) and high variance (0,8). This leads us to (correctly) classify sensor 7 as affected by a calibration fault.

4.2 Attack Classification

To evaluate the proposed methodology under attack scenarios, we injected malicious behavior into the system (the original data did not contain malicious attacks). In the modeled attacks, the malicious nodes try to force the system into a new state (Dynamic Creation) or to delete a valid state (Dynamic Deletion). Malicious behavior is injected in one-third of the available sensors.

In the scenario shown in Fig. 10(a), the malicious attack deletes a correct environment state (29,56) by reporting temperature values lower than other sensors, hence keeping the overall, observed value of the temperature constant at 24. The malicious sensor also keeps the humidity value approximately constant at 70 during that period. (Note that forcing the overall humidity to a precise 70 value would require malicious humidity values greater than 100, which could be easily detected with range checking. In this study, we have decided to maintain malicious values within their admissible range, e.g., [0, 100] for humidity.) As discussed in § 3.4, attacks can be classified by analyzing the observation symbol probability distribution, $\mathbf{B}^{CO}$. Table 6 depicts matrix $\mathbf{B}^{CO}$ for malicious sensor 7. It can be seen that the row probabilities are not orthogonal (see row (29,56) and row (20,71)). According to our methodology, this indicates that an adversary has deleted an environment state (state (29,56) in the example) from the observations. In fact, the considered attack has effectively eliminated a transition from state (20,71) to state (29,56) by forcing

[6]Based on Table 5, we associate model states as follows: (22,80) to (17,88), (17,88) to (13,96), (27,68) to (22,80), and (32,56) to (27,68).



(a) Attack on the temperature readings. (b) Attack on the humidity readings.

Figure 10. Dynamic Deletion attack.

$i \downarrow, j \rightarrow$	(29,56)	(20,71)	(13,78)	(0,95)	(12,94)	(0,0)
(29,56)	0.001	0.999	0	0	0	0
(20,71)	0	1	0	0	0	0
(13,78)	0	0	0.999	0	0	0.001
(0,95)	0	0	0	1	0	0
(12,94)	0	0	0	0	1	0
(0,0)	0	0	0	0	0	1

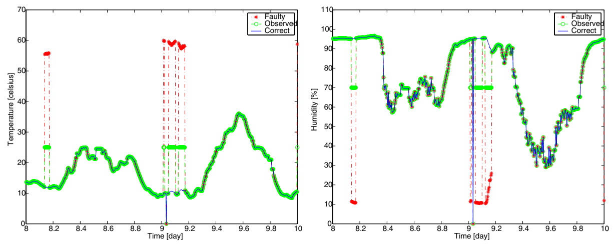
Table 6. B^{CO} matrix for malicious sensor 7 (Dynamic Deletion attack).

the observable environment state to remain (20,71).

A similar injection experiment can be done for Dynamic Creation attacks. In the considered example, malicious nodes inject high temperature values and low humidity values into the system to force a change in the overall, observed state of the environment, whereas the correct environmental temperature and humidity remain approximately constant (see Fig. 11). Table 7 reports observation probability matrix $\mathbf{B}^{CO}$ obtained by the proposed methodology for a malicious sensor 2. It can be seen that the column probabilities are not orthogonal (see column (12,95) and column (25,69)). This indicates that an adversary has created an additional state (state (25,69) in the example).

4.3 Alarm Generation

Figure 12 shows the raw alarms generated for a non-faulty node and for a faulty node at the output of the Alarm Generation module of Fig. 1. We can see that although the gener-



(a) Attack on the temperature readings. (b) Attack on the humidity readings.

Figure 11. Dynamic Creation Attack.

$i \downarrow, j \rightarrow$	(17,86)	(31,56)	(18,78)	(12,95)	(25,69)
(17,86)	1	0	0	0	0
(31,56)	0	1	0	0	0
(18,78)	0	0	1	0	0
(12,95)	0	0	0	0.3546	0.6454

Table 7. B^{CO} matrix for malicious sensor 2 (Dynamic Creation attack).

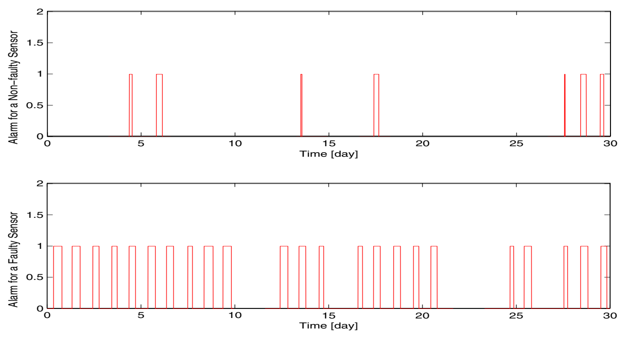


Figure 12. Raw alarms generated for a faulty and a non-faulty node.

ated alarms clearly indicate the absence and the presence of an anomaly, the raw alarm data are quite noisy (e.g., 1.5% false alarm rate for the non-faulty sensor) and appropriate filtering is required to smoothen them.

5 Related Work

Markov models have been widely used in anomaly detection systems [5, 6, 11–14]. In [11], a Markov chain is estimated using a training suite and then used to classify normal versus anomalous behavior by computing different metrics, such as miss probability, miss rate, and local entropy. In [13], a Markov model is used to detect attacks against web-servers and web-based applications. Nong Ye et al. [14] analyze the robustness of a Markov chain-based approach to anomaly detection and conclude that Markov chains perform well only under low noise levels in the data. Hidden Markov Models (HMM) provide a more powerful mathematical tool than standard Markov models and have also been explored in the domain of anomaly detection [5, 6]. In [5], HMMs are used to model an application's behavior by monitoring system call invocations. Similarly to the other anomaly detection approaches cited above, the technique presented in [5] is not designed for distributed systems and does not deal with analyzing and classifying the nature of the detected anomalies.

6. Conclusions

This paper proposes an on-the-fly statistical technique to detect and distinguish faulty data from malicious data in a distributed sensor network. The technique can learn the correct system behavior dynamically with no separate training phase by exploiting the natural redundancy present in sensor networks; furthermore, it can classify faults versus attacks based on structural relations between two types of Hidden Markov Models learned. Whereas the focus of the current study is in sensor networks, the proposed mathematical framework can be generalized for different types of distributed computing environments. For instance, as a future extension of this work we are considering the application of the proposed methodology to monitor intrusions and failures in a large cluster of machines dedicated to running an e-commerce application.

Acknowledgments

This work is supported in part by MURI grant N00014-01-1-0576, the Gigascale Systems Research Center (GSRC/MARCO), and the Motorola Corporation as part of Motorola Center.

References

[1] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. Lessons from a sensor network expedition. In *Proc. of European Workshop on Wireless Sensor Networks*, 2004.

[2] A. D. Wood and J. A. Stankovic. Denial of service in sensor networks. *IEEE Computer*, 35(10):54–62, Oct. 2002.

[3] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. In *First IEEE International Workshop on Sensor Network Protocols and Applications*, pages 113–127, May 2003.

[4] B. Przydatek, D. Song, and A. Perrig. Sia: Secure information aggregation in sensor networks. In *Proc. of SenSys*, 2003.

[5] C. Warrender, S. Forrest, and B. A. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy*, pages 133–145, 1999.

[6] S.-J. Cho and S.-J. Han. Two sophisticated techniques to improve HMM-based intrusion detection systems. *LNCS*, 2820:207–219, 2003.

[7] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pages 88–97, 2002.

[8] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *IEEE Proceedings*, 77(2):257–286, 1989.

[9] M. Basseville and I. Nikiforov. *Detection of abrupt changes: theory and application.* Information and system science series. Prentice Hall, Englewood Cliffs, NJ, 1993.

[10] J. Stiller and G. Radons. Online estimation of hidden Markov models. *IEEE Signal Processing Letters*, 6(8):213–215, 1999.

[11] S. Jha, K. Tan, and R. A. Maxion. Markov Chains, Classifiers, and Intrusion Detection. In *Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, page 206, 2001.

[12] M. Nassehi. Anomaly Detection for Markov Models. Technical Report RZ 3011 (93057), IBM Research Division, Zurich Research Laboratory, 1998.

[13] C. Kruegel and G. Vigna. Anomaly Detection of Web-based Attacks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pages 251–261, 2003.

[14] N. Ye, Y. Zhang, and C. M. Boror. Robustness of the Markov Chain Model for Cyber-Attack Detection. *IEEE Transactions on Reliability*, 53(1), March 2004.