Scheffel and Fröhlich *Journal of the Brazilian Computer Society* (2019) 25:13
https://doi.org/10.1186/s13173-019-0094-6

Journal of the Brazilian Computer Society

RESEARCH Open Access

Increasing sensor reliability through confidence attribution

Roberto M. Scheffel* and Antônio A. Fröhlich

This work is based on an earlier work *WS N Data Confidence Attribution Using Predictors* published in the Proceedings of the 8th Latin-American Symposium on Dependable Computing (LADC) in 2018.

Abstract

The reliability of wireless sensor networks (WSN) is getting increasing importance as this kind of networks are becoming the communication base for many cyber-physical systems (CPS). Such systems rely on sensor data correctness to make decisions; therefore, faulty data can lead such systems to take wrong actions. Errors can be originated by sensor's hardware failures or software bugs and also from the intentional interference of intruders. The gateways that connect such WSN to the Internet are natural intruders' targets as they usually run conventional operating systems and communication protocols. This work proposes a confidence attribution scheme, based on lightweight predictors running on the sensors. The solution also proposes a parameterizable formula, in order to stamp every value sent by a sensor with a *confidence level*, calculated upon the values of a subset of correlated sensors. This work also presents an algorithm that can identify a defective sensor into its subset. The use of predictors and confidence attribution are proposed as the basis of a mechanism that increases the WSN resilience against sensor failure or bad data injection by intruders. Several simulations were performed to evaluate the detection efficiency against different types of sensor errors. This work also analyses mechanisms to deal with concept drifts in the WSN lifetime.

Keywords: Wireless sensors networks, Confidence attribution, Fault detection

Introduction

The advances in the embedded systems technology brought the use of WSN to a variety of application fields. Smart buildings, environmental monitoring, industrial plants, and many other environments can be sensed and controlled by cheap devices, equipped with sensors and actuators, communicating on a wireless network. As these networks are getting integrated into the Internet of Things (IoT), it is essential that they operate in a reliable and trustful manner. Many times, these networks are deployed in harsh environments and are susceptible to interference in sensing and communication. Therefore, fault tolerance mechanisms are essential to ensure correct readings and actuations by the WSN elements. Faults in a WSN can range from incorrect sensor readings, communication failure caused by environmental or intentional interference, to nodes and gateways intrusion by attackers to forge sensor readings or send incorrect commands to actuators.

Although the use of redundant devices increases the fault tolerance in WSNs, consensus or voting algorithms are needed to identify a faulty sensor from the good ones. Several solutions were proposed in the literature, and the majority uses special messages to decide if there was an error and to determine the source. This leads to overhead in terms of latency, bandwidth, and energy consumption in the WSN. Hierarchical or centralized architectures try to minimize this overhead but are subject to errors if the data received is altered by malicious or defective nodes while it is transmitted.

The solution proposed in this work provides self-diagnosis capabilities to the sensor nodes, based on data gathered from correlated neighbors, incurring little communication overhead. To accomplish this, an artificial neural network model is built offline for every type of

*Correspondence: robertoscheffel@utfpr.edu.br
Software/Hardware Integration Lab (LISHA), Computer Science Department, Federal University of Santa Catarina (UFSC), PO Box 476, Campus Trindade, 88040-900 Florianópolis, SC, Brazil

sensors nodes in an interest area, based on the readings from other correlated sensors in the same area. This model is transferred to the sensor. At runtime, each sensor listens to the data transmitted by other nodes in the interest area and uses the model to predict its own value. Comparing the sensed value with the predicted one, each node is able to calculate its confidence or probability of correctness. When the confidence reaches a lower bound, a second step is performed, trying to determine if the node is really faulty or if the discrepancy between the sensed and the predicted value was caused by a faulty neighbor. Every node transmits its sensed value, the predicted value, and the confidence level, to provide information about its current state to the application and to the other correlated nodes. This work is an extension of [1], describing the whole architecture, but only the confidence attribution scheme is completely implemented and evaluated. It extends the study of the parameters' effect on the algorithm's efficiency and makes a verification of how the proposed solution handles different types of errors.

The actual main approaches used to identify faulty nodes in wireless sensor networks are presented in "Related work" section. "Confidence attribution using predictors" section describes the proposed solution, the architecture, and the algorithm used to assign confidence to values read in each node. Afterward, in "Case study" section, the solution is evaluated through a set of experiments on data of a set of real sensors data, and in "Comparison with the hybrid fault detection method" section, a comparison with another algorithm is made using a public dataset. "Concept drift detection" section discusses aspects of concept drift detection on the scenario of correlated sensors models, and "Data confidence as trust mechanism" section describes how the proposed mechanism can be employed to enhance the reliability of data received from the WSN at the gateways or at the servers. Finally, the conclusions and future work are presented in "Conclusions" section.

Related work

A standard solution to increase the dependability of a wireless sensors network is the deployment of redundant sensors to compare readings. It demands a dense deployment of sensors to allow the identification of the faulty sensor among the correct ones. To avoid the dense deployment of sensors, several alternative methods were proposed to verify the correctness of the sensors' data. These methods can be *centralized*, running on the server, *distributed*, running on the sensor nodes, or *hierarchical*, when some particular nodes—like the cluster heads—collect data from a set of nodes and run the diagnosis algorithms [2].

The errors in the sensor's readings were classified by [3] and [4] in four main types. *Outliers* are isolated readings that differ significantly from normal readings expected by the models. The *spike* or *peak* errors are readings that deviate too much from the normal values for a certain period of time. They are composed of at least a few data samples, and not an isolated reading as an outlier. The third type is the *"stuck-at"* error, when the readings present a zero (or very little) variation for a period greater than expected. The amount of time in which the reading has to be "flat" to be considered a "stuck-at" must be determined for each type of sensor. Finally, the *high noise* or *variance* is the occurrence of unusually high variance in the sensor's readings, in such a way that it differentiates it from the usual noise which appears in many sensor types.

Recurrent neural networks (RNN) were used by [5] to predict sensors values, based on previous readings of the sensor and its neighbors. The assumption is that sensors are of the same type, and that difference between the values read by neighbor nodes is bounded by a constant value ϵ. After building and training the RNN, the difference between the sensed value and the predicted value is compared to a threshold η. If the difference is greater than η, the node is considered faulty.

Time series analysis combined with a voting schema is presented by [6], assuming that all nodes sense the same phenomenon, neighbors can communicate directly, and faults occurs interrelatedly. The autoregressive-moving-average —$ARMA(p, q)$— model is applied, with p autoregressive terms and q moving-average terms. The calculation of the regression formula's parameters is done on readings that are known to be correct, before the sensors' deployment. In the voting phase, the readings of the neighbors are collected. Then, the median of these readings is calculated and compared with the actual reading of the sensor. If the difference is greater than a threshold τ, the read value is considered faulty. Faulty values are not included in the node's history, to not disturb the moving average used in classification.

Statistical analysis of sensed values is another method applied for fault detection. The distributed fault detection (DFD) algorithm, proposed by [7], uses the statistical technique of group testing (GT) that identifies a small number of defective items in a large population. Sensors are supposed to be uniformly and independently distributed in a 2D space. A sensor is considered defective if its readings differ significantly from other sensors readings. All nodes exchange their readings with the neighbors, run an *outlier test*, and broadcast the result. These steps are repeated for L rounds. In the last phase, every node updates its status, based on the results obtained from data exchanged in the previous rounds, comparing to on a threshold γ. The values of L and γ determine the trade-off between *false alarms* and *no detection* and are hard to determine for large networks. The authors also proposed an adaptive algorithm that dynamically determines the values of

L and γ. A modified three-sigma edit test is proposed by [8, 9], using the ratio between the current value, the median and the normalized median absolute deviation of the last n readings. If this ratio is greater than 3, it considers the reading an outlier and the sensor as faulty.

The distributed Bayesian algorithm (DBA) is presented by [10]. Sensors of the same type calculate the probability of being faulty in three steps. First, the nodes periodically exchange their values and probabilities to the other nodes in radio range R. Sensors are in the same state (*faulty* or *not faulty*) if the difference between their readings is smaller than a specified threshold r_t. The Bayesian formula is used to calculate the fault probability of each node. In the second step, adjustments are made to avoid that a good node surrounded by faulty nodes becomes faulty (and vice versa). In the third step, nodes with a fault probability higher than a given threshold τ will send a warning message to the sink.

A distributed fault detection based on hidden Markov model is presented in [11]. Each node uses the difference between its value and the values of the neighbors, determining its state as *possibly normal* or *possibly faulty*. In the sequence, the probability of the node to change its state from *good* to *faulty* (or vice versa) is calculated using a transition matrix built from the results obtained in the first step. The algorithm assumes a WSN composed by a dense deployment of sensors of the same type, to directly compare readings.

A fault-tolerant algorithm for event detection in WSNs is proposed in [12], called spatiotemporal correlation based fault-tolerant event detection (STFTED). The proposed scheme uses a location-based weighted voting scheme (LWVS). It explores the spatiotemporal correlation between sensor nodes, assuming a dense deployment of sensors of the same type, to detect events. It also assumes a mean value m_n representing a normal reading (or absence of event) and a mean value m_e representing the presence of an event. At the node level, the readings of the neighbors are weighted based on their distance. Closer nodes have a higher influence in the estimation function, and vice versa. In this step (LWVS), an estimator R_n for the state node n is calculated, which can be inaccurate. So, the second step (STFTED) uses the Bayesian formula to calculate the probability of a node to be faulty, based on the estimation of other nodes in the same *fault range*. If the majority of nodes have a high likelihood of normal readings, abnormal readings are considered faulty, and the contrary is also true.

The authors of [13] proposed an algorithm named fault detection scheme (FDS), also based on a local step, carried out on sensor nodes and a second decision step that runs on the cluster head nodes. On the local step, each node calculates the probability of node i be faulty (joint probability or PJ_i), using a Bayesian network that uses the energy level and the sensed data of node i (EL_i and SD_i). If the probability of being faulty exceeds a threshold δ, the node classifies itself as *possibly faulty* (PF). Otherwise, it classifies itself as *possibly normal* (PN). Each node sends PJ_i and its decision to the cluster head (CH), which executes the second step of the scheme. The CH maintains a table called probability join table (PJT) with the PJ of every node from the cluster. For each node i, the PJT contains the previous and the actual PJ_i. When the node decision is PF and the difference between PJ_i^{t+1}, and PJ_i^t is greater than a threshold γ_2, then the node is considered faulty. Otherwise, it is considered a false alarm. Based on FDS, the authors proposed a distributed fault-tolerant algorithm (DFTA) in [14], where they describe a scheme to make the elimination and the recovery of faulty nodes.

A method based on logistic regression is proposed in [15], with the model construction (called Learning Step) executed on the sink, processing data from all sensors. After training the model, it is sent to the nodes, where it is executed. The value predicted by the model is compared to the one read by the sensor. If the difference is greater than a threshold, the node is classified as faulty; otherwise, it is classified as normal. No method to determine the threshold value is proposed, and the authors decide it "based mainly on experience and intuition."

In [16], a distributed fault detection for WSNs in smart grids is presented, based on credibility and cooperation among sensors. Each sensor evaluates its status as suspicious or not, based on the mean and the variance of a window containing the last k-sensed values. A healthy sensor keeps the variance bounded. When a sensor detects itself as suspicious, a *diagnostic request* is sent to the neighbors, and the *diagnostic response* messages are used to determine the node's state. After receiving the response of sensors in an area determined by a radius R, the node can update its probability of being healthy or faulty.

Neural networks are used in [17] to detect and classify different faults in a WSN of homogeneous sensors. The work assumes that the sensors are uniformly distributed and with a set of anchor nodes (cluster heads) that have broad radio range and are fault free. A genetic algorithm combined with gradient descent is used to train the neural networks. These neural networks classify the state of the nodes. After node classification as secure or faulty, the last ones are disconnected from routing paths.

The use of large models, with several inputs and complex interconnections, may not be the best choice for prediction models. When data is sparse or with complex interactions, the use of ensembles can obtain better prediction results. As each classifier explores specific competence domains, an ensemble outperforms single classifier that tries to handle all the inputs in a unique algorithm [18]. In their work, [19] used several classifiers to determine if the readings of WSN are faulty or normal. The

classifier uses the readings of a sensor and from the neighbor nodes as input. All classifiers run on the gateway, in a centralized way. The outputs of the classifiers are then joined in a single decision and an estimated value, using weighted majority voting.

Most of the presented solutions use a dense deployment of sensors of the same type, comparing values that should be equal or very similar. The main difference between them is the prediction model, and some of them use extra packet exchange in voting rounds [13] or for value testing [7]. Although it is a possible and realistic WSN configuration, these models are not applicable when sensing a variety of values (e.g., environmental monitoring). Replication is essential to ensure reliability, but the presented solutions demand dense deployment of *each* type of sensors. This raises costs and causes communication interference and network overload. The proposed solutions also rely on determining models and parameters *before* the WSN deployment, not considering future environmental changes. Extra packet transmissions in *diagnostic phases* as in [16], or exchanging test results like in [7], causes communication overhead and introduce delays every time a node's failure occurs.

The fault detection scheme proposed in [13] and [14] uses a hybrid algorithm, with a step carried out on the node, and another performed on the cluster head, which has higher processing and memory capabilities. In opposition to [17], the proposal of this paper is not to automatically disconnect faulty nodes, but allow each node to determine its confidence on the sensed value. This can be used to identify a faulty node, as well as to detect data corruption by intermediate nodes. Nevertheless, the application can also cancel the interest on faulty nodes, making them stop sensing until they can get maintenance.

The objective of the local confidence determination of the sensor read value proposed in this work has as objective to avoid extra messages exchange between the neighbor nodes to verify faults and identify their origin. It also provides a local algorithm that can attest the data reliability to make local decisions safer. Finally, it provides a security increment, as an intruder cannot forge data without getting the exact model and input parameters involved in the confidence attribution process.

Confidence attribution using predictors

Decentralized fault detection approaches try to minimize the message and time overhead inherent to a centralized approach but have to deal with limited input sets and less computational power to perform its work. Models used to perform fault detection at the sensor level have to take into account the resource constraints. As radio communication is the most power-consuming resource on a WSN and packet collision is a problem in WSNs with a large number of sensors, or with high sampling rates, diagnosis messages for fault detection should be avoided.

The fault detection techniques can be applied at different levels and classified in several ways. When considering the parties involved in the process, they can be classified into three different classes. In a *self-diagnosis* method, the node itself can identify faults on its components. In a *group detection* mode, each node monitors the behavior of other nodes to detect errors. When using a *hierarchical detection*, the detection is shifted to more powerful nodes, such as cluster heads or the gateway. In the WSN perspective, the ideal fault detection method would be distributed and use a self-diagnosis approach. A group detection approach can also be used, but in a *"speculative mode"*, without the use of specific diagnosis or control messages.

The assumption made in this work is that the network is composed of different types of sensors, monitoring several aspects of the same phenomenon or different interconnected phenomena. So, it can be assumed that each node produces data correlated with the data produced by some other nodes. The network has to be designed in a way that in different interest areas is a set of sensors that produce correlated values. The size of this set is not fixed and can vary concerning the correlation between the sensed physical quantities. Another assumption is that every node can read the data transmitted by the other nodes. If the communication is ciphered, some global or group key schema has to be used. Messages can be encrypted with a group key, allowing the data reading by other authenticated nodes. For authenticity, the messages can be signed with the node's private key, avoiding the message content from being altered by other nodes.

The proposed scheme aims to provide nodes with self-diagnosis capabilities, based on data gathered from correlated neighbors, with no communication overhead. For every node type in an *interest area*, a predictor model is built offline, based on past readings from the different types of sensors in this area. This work uses artificial neural networks, but any other predictor could be used. The only requirement is that the model has to be easily transmitted over the network and not demand much memory or processing capabilities. This model (or the new parameters, when updating a model) is then transmitted to the sensor, where it is stored and executed.

At runtime, each sensor listens to the data transmitted by other nodes and uses the model to predict its value. Each node calculates its confidence—or probability of correctness—comparing the sensed value with the predicted one. When the confidence reaches a lower bound, a second step is performed trying to determine the source of the fault. This step is necessary to define if the cause of the discrepancy is an erroneous reading, or if it is caused by an erroneous input from another sensor. To accomplish this, every node transmits the read value, the predicted value,

and the confidence level, to provide information about its current state to the application and the other nodes. There is no extra packet transmission, only an increment in the size of the transmitted packet. Assuming a 64-bit value as a sensor reading, the increase is 9 bytes: the predicted value plus one byte for the confidence (a value between 0 and 100). If more confidence levels are needed, more bytes can be added to the packet. It is a design decision whether more granularity in the confidence levels is worth the increment on the network packet size.

In several monitored environments, the measured values can change in range, and their correlations may vary over time, characterizing many WSNs as *non-stationary environments*. The authors of [20] present a process flow for anomaly detection under such conditions, composed of five sub-processes, enumerated below:

1. *Change detection*: achieved through data monitoring to detect changes in the data distribution. If a significant change occurs, then a model update must be performed.
2. *Training set formation*: the data vector for model construction and training is formed using sliding windows techniques, discarding n oldest samples and adding n new samples to the training set.
3. *Model selection*: the optimal parameter set is determined for the new training dataset.
4. *Model construction*: with the parameters determined in the previous step, a new model is constructed. It can be done in *batch mode*, where the $model(t-1)$ is discarded and the new $model(t)$ is built from scratch, or in an *incremental mode*, where $model(t-1)$ and n new data vectors are used to build $model(t)$.
5. *Anomaly detection*: the new $model(t)$ is used as the new anomaly detector for fresh data $X_{t+1}, X_{t+2}, \ldots$.

The learning framework proposed by [21] shown in Fig. 1 applies to such environments. When a change is detected, the model is updated and sent to the classifier. When applying this framework to WSNs, the *feature extraction*, *change detector*, and *adaptation* processes can run on a dedicated server or the Cloud, in a centralized approach. Once the model is built or updated, it can be transmitted to the WSN nodes and used to assign confidence to their readings. It can be the sensor node itself or intermediate nodes, in a hierarchical architecture.

The first step is the model building for a specific sensor, based on the historical data from the sensors of an interest region. This implies that a *new* WSN will not have a reliable predictor on its first deployment. After some time running, the data collected by the sensors can be used to train a primary model for each sensor and deliver it to the sensors. Afterwards, an update can be performed every time the *adaptation* process achieves a more accurate model.

The feature selection process searches the smallest input set for a predictor that produces the best results concerning accuracy and model size. This process can be resource and time-consuming, as many combinations of the inputs have to be evaluated. For large input datasets, or when the correlation between the variables is unknown, automatic attribute selection techniques can be employed. There are several algorithms for this, with different approaches. These algorithms can be grouped into three classes [22]. *Filter methods* use statistical measures to assign a score to each feature and create a ranking. The lower ranked features are removed from the dataset. The methods are often univariate and consider each feature independently, or about the dependent variable. Some examples of some filter methods include the chi-squared test, information gain, and correlation coefficient scores. *Wrapper methods* consider the selection of the feature set as a search problem. Different combinations are prepared, evaluated, and compared to each other. A predictive model is used to evaluate a combination of features and assign a score based on model accuracy. The search process may be methodical as a best-first search, stochastic as a random hill-climbing algorithm, or heuristic-based like forward and backward passes to add and remove features. An example of a wrapper method is the recursive feature elimination algorithm. *Embedded methods* learn which features increase the model accuracy while it is created. The most common type of embedded feature selection methods are the regularization methods. Regularization methods—also called penalization methods—introduce additional constraints into the optimization of a predictive algorithm (such as a regression algorithm) that bias the model toward lower complexity (fewer coefficients). Examples of regularization algorithms are the LASSO, elastic net, and ridge regression.

There are many machine learning tools available that can efficiently execute this task, such as Weka [23] tool or scikit-learn [24] machine learning toolkit. Different techniques for feature selection can be used, such as a filter method or a wrapper method. As communication is costly in WSNs, it is necessary to impose an extra restriction: the input set must be selected from the sensors close to the target sensor. The notion of *close* can vary, based on the communication layer. In a single-gateway architecture, it can denote nodes inside radio range, so the node can listen to their transmissions. In multi-gateway architectures, *close* can mean nodes two or three hops away, once the routing makes their packets be re-transmitted by a neighbor that can be listened. In this context, it is a node whose data can be observed by the current node, which needs the values as its predictor input. This also implies that the application has to know the location of the sensors, which is a requirement of many routing protocols. Figure 2 illustrates the whole process, where *feature extraction* and

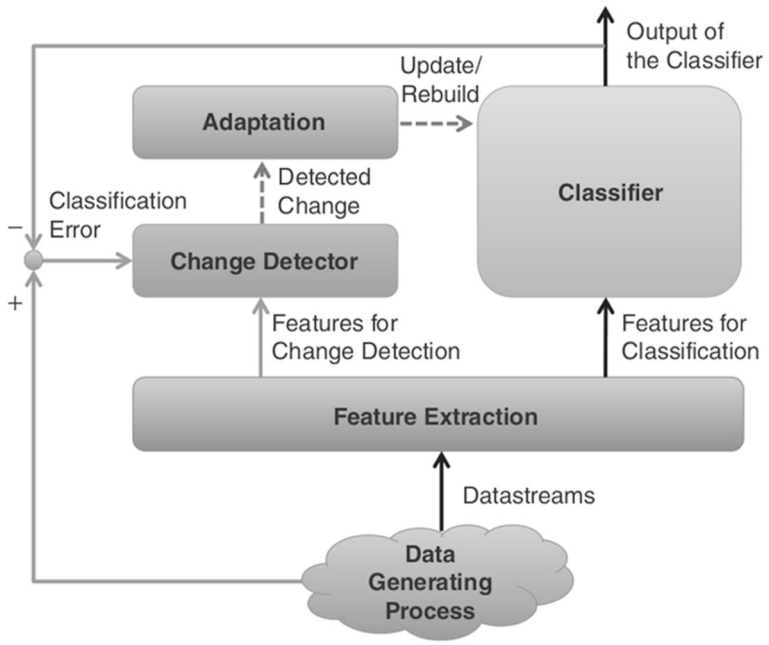


Fig. 1 Learning in non-stationary environments [21]

model building and training are performed offline, using historical data from a database.

After identifying the set of sensors that are more correlated to the target sensor, a model for each node was built automatically. In the first experiments, multilayer perceptrons were used. The number of layers and neurons in each layer can be determined by heuristics and by rules. The input layer has one neuron for each input value. An extra input can be used for backpropagation or bias. The output layer has one neuron, corresponding to the predicted value. Rules like the proposed by [25] can determine the number of neurons in the hidden layer. Pruning

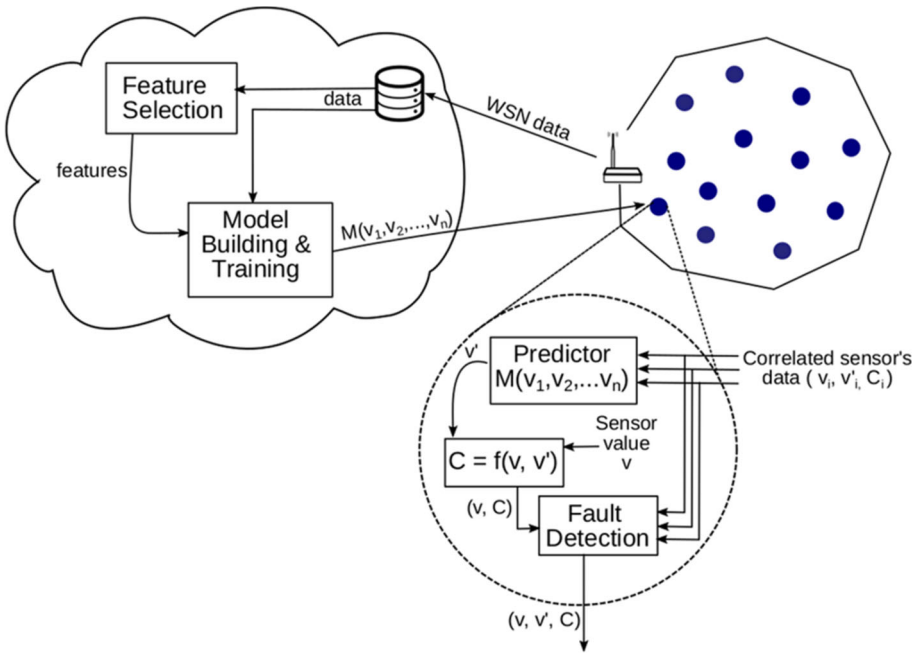


Fig. 2 Model learning and confidence attribution scheme

methods [26] can determine the optimal number of neurons in the hidden layer. Once built and trained, the models are transmitted to the respective nodes.

When the predictor is received, sensors can start to determine the correctness of their readings. The *confidence level* C of a node is a function that evaluates the difference between the predicted value $\hat{v}$ and the sensed value v, so $C = f(v, \hat{v})$, with f as described in Eq. 1. In this work, the *mean absolute error* (see Eq. 2) obtained in the training process is used to calculate the result of the function. Comparing it to the root square mean error (RSME), the authors of [27] state that MAE is a natural measure of average error and is unambiguous, being widely used for model-performance evaluation.

$$f(v, \hat{v}) = \begin{cases} 1, & if \ |v - \hat{v}| \leq \beta \times MAE \\ 1 - \frac{|v-\hat{v}|-\beta \times MAE}{\alpha \times MAE}, & \text{otherwise} \end{cases} \quad (1)$$

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |v_i - \hat{v}_i| \quad (2)$$

The constants α and β adjust the sensitivity of the function 1. The value of β defines the tolerance margin (relative to MAE) that considers a value correct with 100% confidence. If the model has good accuracy, a small value of β can be used to detect small variations in the readings. Otherwise, if the monitored value presents high variability, this constant has to get a larger value in order to accommodate the variations. The value of α defines the "*velocity*" that the confidence decreases as the difference between the predicted and the monitored value increases. The smaller α, the faster the confidence decreases. If the predicted and monitored values are too distant from each other, the function may result negative, in which case the confidence assumes 0 (zero).

The prediction models may show circular references (the model to predict variable S_1 depends on variable S_2, and the model to predict S_2 depends on S_1), since the correlation is reflexive. The monitoring applications of WSN perform periodic sampling of the sensor values. So, we assume that it is possible to determine a period P, in which at last one reading of each sensor type is transmitted to the application. The majority of WSN communication protocols do not guarantee the order between messages while they are retransmitted by the intermediate nodes. Also, the exact instant when a message arrives on a node is not determined only by the instant it is produced. It is also highly influenced by communication delays and network load. The protocols only guarantee that data is delivered at each period P, without ordering guarantees. So, it is only possible to assume that the data from the previous period is available. Therefore, models are trained using data from the last period to predict the values of the actual period. Each node will passively listen to the data sent by other nodes, collecting the inputs for its predictor. Data produced by the nearest sensor of the input types of its model will be buffered. This data will then be used as input values of the model for the local node, in the next period $t+1$ (Eq. 3).

$$\hat{v}_i^{t+1} = f\left(v_1^t, v_2^t, v_3^t, \ldots, v_n^t\right), \text{for all } n \text{ inputs} \quad (3)$$

As the environment changes, the *change detector* process presented in Fig. 1 has to detect the changes and trigger the *adaptation* process. Change detection methods can be grouped into four main families [21]:

- Hypothesis test uses statistical techniques to verify the classification error of a fixed-length set of readings. The variation of the classification error is compared to the error of the training dataset.
- Change-point methods also uses a fixed-length data sequence, analyzing all partitions of the data sequence to identify the instant when the data changes its statistical behavior, called change-point. The main drawback of this method is the high computational complexity.
- Sequential hypothesis test instead of analyzing a fixed-length window of data, this method inspects each incoming sample, until they have enough evidence that a change has occurred or not.
- Change detection tests are specifically designed to analyze the statistical behavior of data streams sequentially. Most of them operate by comparing the prediction of absolute error or its variance to a specified threshold. The threshold is hard to determine at design time. Some adaptive algorithms were proposed, using cumulative errors.

The authors also propose the use of hybrid change detection. A change detection test can be used in a first layer, followed by a validation layer that uses a change-point method.

Model update, performed after a change is detected, consists in retraining the model with new data. The main approaches are *windowing*, *weighting*, and *random sampling*. At this step, one of the main questions is if the model has to forget the oldest rules and reinforce the new ones, or if the learning has to be cumulative. In the former case, the model is entirely retrained using new data. This implies retraining the model every time a change is detected but can lead to smaller models. The latter case, incremental learning can keep past knowledge, but models tend to be larger, demanding more memory and processing.

In our proposed solution, a newly deployed WSN has to run without confidence attribution for some time, to

gather enough data to build and train models for the different sensors. Models are then built for each type of sensor in the same region and can be sent to them in a *group communication*. The application can control the model transmission frequency to avoid network flooding. Model updates are expected only when the correlation between the sensed values changed. In a *training phase*, some overhead will happen when some models have to be updated. After some time—that depends on the environment dynamics—it is expected to occur very rarely. The compressed size of the model (around 3 KB) makes it feasible to be transmitted over the WSN. The proposed solution does not fit well to environments where the correlations are changing continuously at a high rate. The use of cumulative learning, although producing larger models, was chosen in the first experiments as it is expected that environments with cyclic changes will *stabilize* over time in the learning process, demanding much less (or even none) updates after running for a period. The different approaches for drift detection and model retraining are still under evaluation and a discussion about it is made in "Concept drift detection" section.

Fault Identification and the interference problem

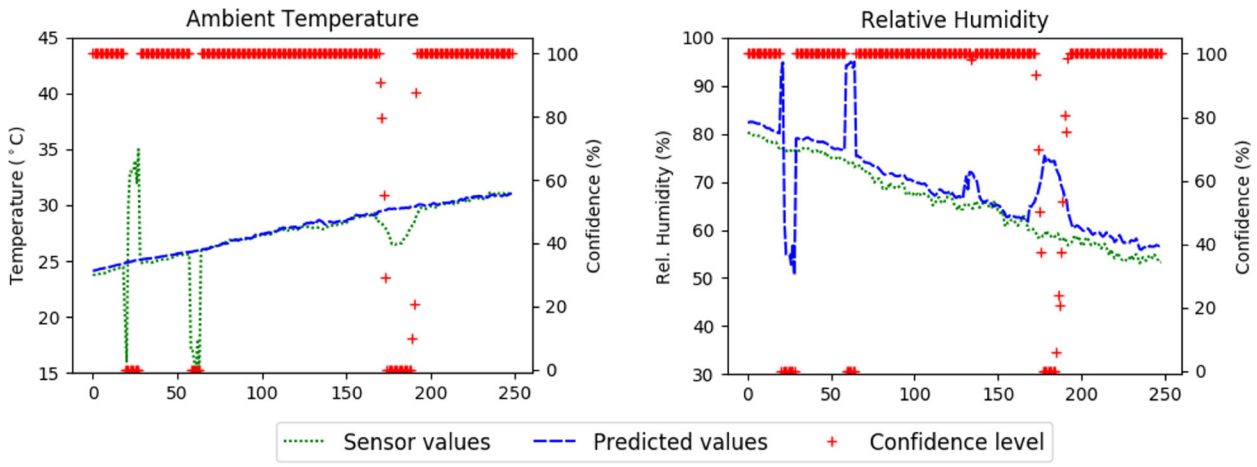
If each node sends only the value of the sensor and its confidence, it would be hard for other nodes to make a reasonable statement about its confidence on the sensed value. It happens because a wrong input in the model causes distortions in the output, leading to an erroneous confidence attribution. For this reason, fault detection algorithms tend to use voting or joint probability tables to identify the fault source. In the scheme proposed in this work, every node sends three values: the sensed value $\boldsymbol{v}$, the predicted value $\hat{v}$, and the confidence level $\boldsymbol{C}$. Knowing these three values from the correlated neighbors, every node is able to autonomously verify the correctness of its readings, without extra messages. On receiving an input value v with low confidence, the local predictor will use the predicted input value $\hat{v}$ to calculate its prediction and confidence for local v. Doing so, the failure of a sensor will take longer to affect other sensors' confidence by interfering in their prediction models. It also enables the identification of the faulty node, as it will show lower confidence, unlike the other (well functioning) sensors. At each node, the algorithm that performs the process of confidence attribution shown in Fig. 2 is described in Algorithm 1.

Applying only the Eq. 1 to the sensed values of the correlated sensors will cause an error spreading as illustrated in Fig. 3a, leading to an *interference problem*. As the reading faults occur in the first sensor (ambient temperature), the predicted values of the second sensor (relative humidity) change proportionally, but in the opposite direction, once they show an inverse correlation. If no information

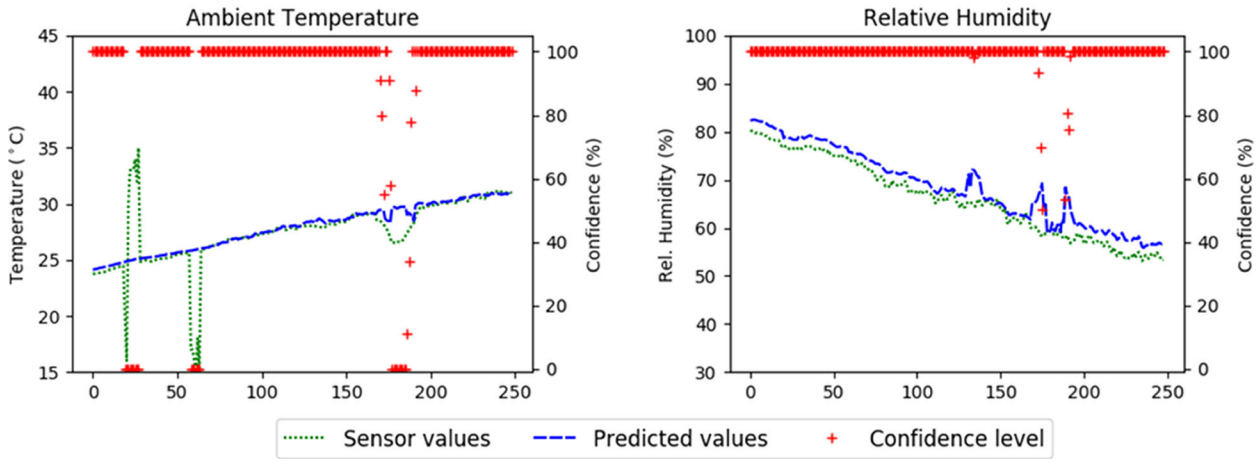
Algorithm 1 Confidence Attribution Routine

```
1:  procedure Confidence_Attribution
2:      // Calculates the predicted value based on sensed values
3:      v̂_o ← model(values)
4:      diff ← |y − v̂_o|
5:      if diff ≤ (β ∗ MAE) then
6:          conf_o ← 100
7:      else
8:          conf_o ← (|y − v̂_o| − β × MAE)/(α × MAE) × 100
9:          if conf_o < 0 then
10:             conf_o ← 0
11:         end if
12:     end if
13:     // Changes low confidence readings to the predicted values
14:     for each c_i ∈ confidences do
15:         if (c_i < γ) then
16:             values[ i] ← predictions[ i]
17:         end if
18:     end for
19:     v̂_p ← model(values)
20:     diff ← |y − v̂_p|
21:     if diff ≤ (β ∗ MAE) then
22:         conf_p ← 100
23:     else
24:         conf_p ← (|y − v̂_p| − β × MAE)/(α × MAE) × 100
25:         if conf_p < 0 then
26:             conf_p ← 0
27:         end if
28:     end if
29:     if (conf_o > conf_p) then
30:         return (v, v̂_o, conf_o)
31:     else
32:         return (v, v̂_p, conf_p)
33:     end if
34: end procedure
```

but the read value is available, there is no way to determine if the local value is good or defective. When the nodes transmit the sensed value, the predicted value, and the confidence level, Algorithm 1 is able to verify that relative humidity sensor values have greater confidence when the predictor uses the ambient temperature sensor's predicted value as input. Doing so, the confidence in its readings remains high, and the faulty readings of the damaged (or malicious) sensor do not cause harmful interference. In the illustrated example, the γ value is set to 40, replacing the sensed value of input by its predicted value if the confidence is lower than 40%. This is shown in the graphs of Fig. 3b. The three first peaks (between time 15 and 70) on the first sensor caused no interference in the correlated sensors' predictions, keeping the confidence of the correct sensor almost unchanged. Between time 150 and 200, the predicted value starts to slowly deviate from the sensed value, making the confidence decrease as the input error increases. When the confidence goes below 40%, the predictor begins to replace the value from the erroneous input by its prediction, which restores the sensor's confidence level. It is important to note that the *ambient temperature* graphs shows a subtle but important difference in Fig. 3b. In the lower, the interference from the *relative humidity* input is reflected on the input between time 150 and 200, until the algorithm decides to replace the sensed values by the predictions.



(a) Confidence Attribution only by sensed values



(b) Confidence Attribution replacing low confidence input by it's predicted value.

Fig. 3 Fault isolation by Algorithm 1. **a** The predictions of Relative Humidity are distorted by ambient temperature. **b** The algorithm isolates the distortion when confidence goes bellow γ

In the next section, a case study is presented, illustrating the whole process and making an analysis of the impact of different values of the algorithm's parameters (α, β, and γ). Despite using artificial neural networks and the previously presented confidence formula (1), the proposed solution can be changed in several ways. The central point of the architecture is the use of a *predictor*, a *confidence attribution formula*, and the use of the output of other sensors (read value, predicted value, and confidence) to determine origin of a fault. Any component can be replaced. Another type of predictor or a specialized confidence formula can be employed, without significant impact on the proposed schema.

The main reason use of a predictor and a confidence attribution formula instead of a classifier is because the objective is to be able to detect the different levels of variation on the read values, not just classify the values as *correct* or *incorrect* (and some intermediary classes). The confidence level can be adjusted to a large number of levels, only by adjusting the number of bytes used to represent it, determined by the application's needs, by the sensors' processing capabilities or by the number of bytes available in the network packets.

Case study

To validate the proposed confidence attribution scheme in a realistic scenario, we run simulations using real WSN data along with the scikit-learn [24], Theano [28], and Keras [29] frameworks. The data consists of nine time series, each 1-year long and with a temporal resolution of 1 min. The series were acquired from an environmental monitoring WSN installed on a partner's solar

power plant and subsequently cleaned and validated by the engineers that commissioned the plant. Segments corresponding to 5% of each time series were used to perform feature selection with the *SelectKBest* selector from the scikit-learn framework.

For each variable, the six other variables with the highest correlation were selected to be used as the predictor's input. The variables and the selected features are shown in Table 1. The hour and minute of each datapoint's timestamp are recurring features that are relevant to all variables. The month of the year, which would have been a good indicator of the seasonality of solar energy generation, was not detected as a feature because the training data covered less than 1 month. This will intentionally make the model susceptible to concept drifting, and it will be shown and discussed in "Concept drift detection" section.

After the feature selection, an artificial neural network (ANN) was built for every sensor, using the multilayer perceptron architecture. All ANNs have the same structure, with six neurons in the input layer (one for each feature), five neurons in the hidden layer, and one neuron in the output layer. The Keras API [29] and the Theano library [28] were used to build, train, and evaluate the ANNs. The small size of the model makes it suitable to be transmitted to the sensor nodes and executed locally to make the predictions. After the ANNs were built, the MAE was calculated over the training dataset for each variable, as shown in Table 2. As an example, the MAE value for the ambient temperature makes a value of ν be considered having 100% confidence if it lies in the range $[\hat{\nu} - 0.41697, \hat{\nu} + 0.41697]$, where $\hat{\nu}$ is the predicted value, when $\beta = 1.0$. This interval can be shrunk or stretched by modifying the value of β.

Once the model building phase was finished, several simulations were carried out to verify the accuracy of the proposed confidence attribution scheme. Five different data chunks were extracted from the original dataset, each one with 1440 samples (1-day sampling). The evaluation was done in two steps. First, to evaluate the models efficiency in identifying and isolating errors in one sequence (its origin), several errors of the *outlier* type [4] were injected at random points in the data in the *ambient temperature* and the *Datalog internal temperature* sequences. One hundred forty-four errors were injected in each sequence, independently, and 144 errors were injected simultaneously on both sequences. The objective of injecting simultaneous errors in both sequences was to verify if the substitution of read value by the predicted one in a sequence would lead Algorithm 1 to mask the error on the other sequence. Algorithm 1 was executed on the data chunks, measuring the detection and false positives rates. The variation of the three algorithm parameters—-namely β, α, and γ—allowed to verify their influence on the algorithm efficiency. In the next set of experiments, the other three types of error defined by [4]—*peaks*, *"stuck-at,"* and *noise*—were injected in the data sequences, to verify the algorithm sensitiveness to each type of error. The objective was to verify if errors are correctly detected when the values deviate significantly from the normal ones. No error classification is expected in this work. So, if different types of error appear, causing significant deviation on the monitored values, the algorithm only assigns a lower confidence to the variable, no matter what kind of error has occurred.

Table 1 Feature selection result

Target	Selected predictor inputs
Diffuse solar irradiation	Direct solar irradiation, global solar irradiation, barometric pressure, Datalog internal temperature
Direct solar irradiation	Diffuse solar irradiation, global solar irradiation, ambient temperature, Datalog internal temperature
Global solar irradiation	Diffuse solar irradiation, direct solar irradiation, barometric pressure, Datalog internal temperature
Barometric pressure	Diffuse solar irradiation, global solar irradiation, ambient temperature, Datalog internal temperature
Rainfall Index	Diffuse solar irradiation, barometric pressure, ambient temperature, Datalog internal temperature
Wind direction	Diffuse solar Irradiation, barometric pressure, ambient temperature, Datalog internal temperature
Ambient temperature	Global solar Irradiation, barometric pressure, Datalog internal temperature, relative humidity
Datalog internal temperature	Global solar irradiation, barometric pressure, ambient temperature, relative humidity
Relative humidity	Global solar irradiation, barometric pressure, ambient temperature, Datalog internal temperature

Table 2 MAEs calculated for predictors

Sensor	Model's MAE
Diffuse solar irradiation	2.5799
Direct solar irradiation	15.8076
Global solar irradiation	16.2823
Barometric pressure	0.7605
Rainfall Index	0.0017
Wind direction	28.9098
Ambient temperature	0.41697
Datalog internal temperature	0.7318
Relative humidity	3.3265

Figure 4 shows the effects of injecting outliers in the sequences of the *ambient temperature* and the *Datalog internal temperature* values. The peaks in the values (green dotted lines) on the two top graphs show the points where the errors were injected, at random instants. The peaks in the predicted values (blue dashed lines) show the effects of the injected errors in the predictor's output of the related variables. When the incorrect input leads to confidence lower than the threshold defined by γ (40% in this experiment), the algorithm makes the substitution by the input's predicted value, trying to get the predicted value closer to the range of correct values if its value is correct. So, it is normal that small peaks appear on the predicted values.

Algorithm parameter evaluation

To evaluate how the algorithm parameters influence on the results, several errors of *outlier* type were randomly injected in each data chunk. The amplitfude of the error was also randomly defined. So, it is expected that not *every* error is detected with confidence bellow γ, as some generated values can fall inside the range of acceptable values. The ambient temperature and the Datalog internal temperature sequences were chosen to have their values changed, as they are input for all other predictors, and also show high interdependence. The errors were injected as described previously. The process was repeated ten times with each data chunk, and the mean values of the detection and false positive rates were calculated.

The *error detection rate* (EDR) is the percentage of the injected errors that are correctly identified by the proposed algorithm. A *false positive* occurs when a node with no error injection presents a confidence level below the γ parameter (Algorithm 1, line 15), at the next period where an error was injected in another sequence. As explained in formula 3, each read value is used as input of the predictors in the next time period. This implies that low confidence on this variable results from a wrong prediction

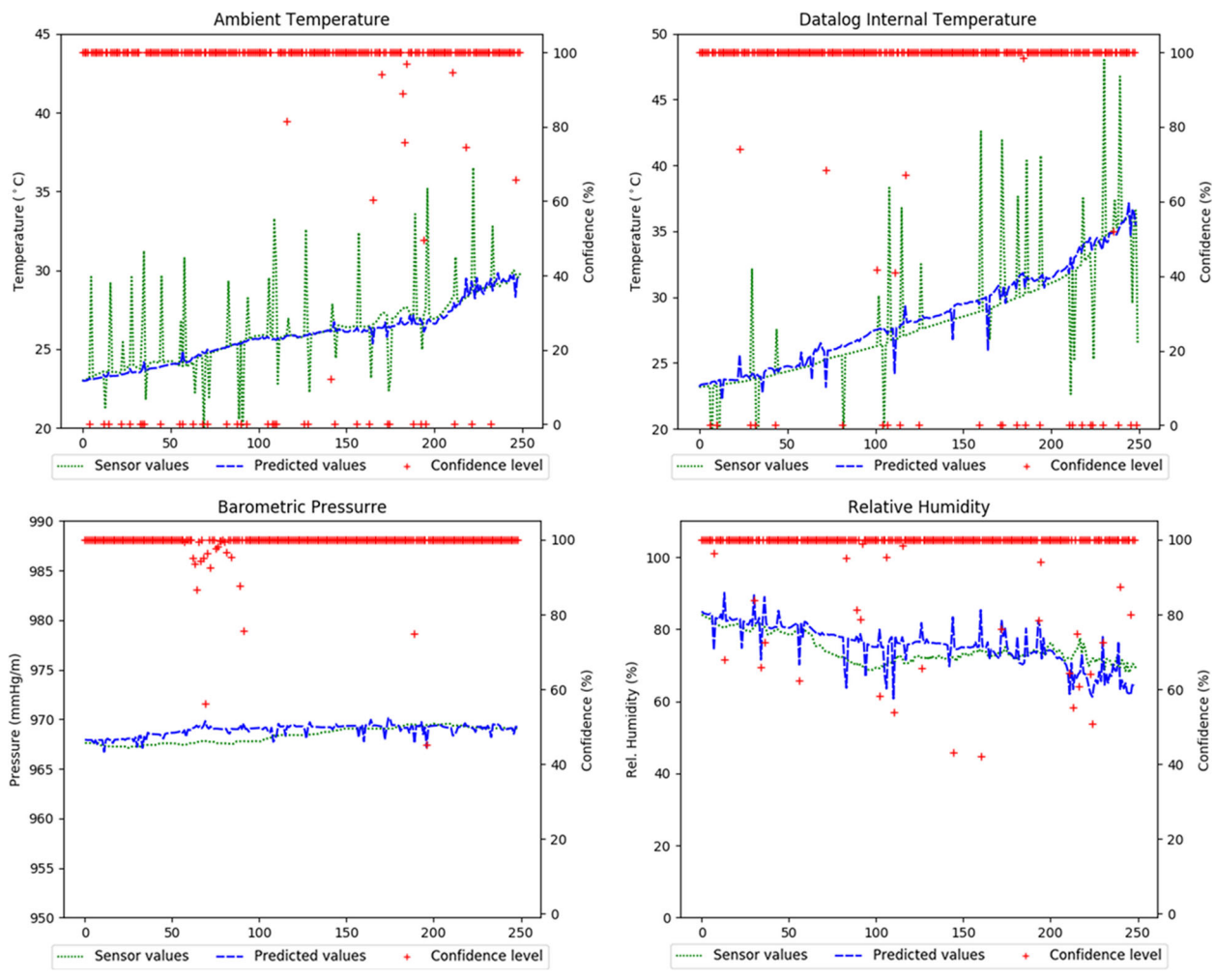


Fig. 4 Error injection for evaluation of Algorithm 1

caused by the injected error. This is unexpected and the algorithm tries to eliminate, or at least to minimize, its impact. As some errors may occur in the sample data—as it is data from real sensors—the confidence of all variables *before* the error injection was calculated and stored. For each low confidence found *after* the error injection, it was compared with the recorded value. So, false positives were counted only when ***new*** errors were detected. The *false positive rate* (FPR) is calculated based on the number of injected errors, making it possible compare the results of experiments made with different sizes if input set and different numbers of injected errors. The formulas are $EDR = (detected\_errors \times 100)/injected\_errors$ and $FPR = (false\_positives \times 100)/injected\_errors$.

The experiments results are shown on Fig. 5, with the minimum confidence (γ) varying from 20 to 90 in intervals of 10, and α and β varying from 1.0 to 4.0 in a step of 0.5, making a total of 392 evaluated combinations. As explained before, a small value of β means that the current reading can show only a little discrepancy from the predicted one to be considered 100% correct. The opposite is true for larger values of β. The α parameter determines the width of a range of values where the confidence decreases from 100 to 0%. The wider the range, the slower the confidence decreases. So, small values of α make the algorithm drop confidence very fast, and vice versa. Finally, the γ parameter defines the confidence level below which the algorithm tries to replace the read values by the predicted ones, searching for a possible error. The lower the value of this parameter is, the more *"lenient"* the algorithm is with the investigation of possible errors.

The objective is to maximize the number of detected errors and, at the same time, to minimize the number of false positives injected. So, the best combination of parameters will be the one that gets the biggest difference between these two results. Figure 5 shows the EDR and FPR values obtained for every combination of the three parameters. The highest EDR were obtained using the most restrictive combination of parameters ($\beta = 1.0$, $\alpha = 1.0$, and $\gamma = 90$), detecting 94.2% of the injected errors. But this combination also shows the highest FPR, as high as 79.5% of the injected errors. This is expected because any reading that deviates 1.1 times the MAE from the model will be considered an error and tested against the predicted value. The FPR decreases for smaller values of γ. It also decreases as the values of α and β increase, with FPR getting close to zero when both parameters are 4.0. The graphs of Fig. 5 show that average values for all parameters show balanced results.

As the parameters can be individually adjusted for different scenarios and application needs, a reasonable start point is setting α and β parameters to a value around 2.5 and γ to 50, and adjusting them until the desired detection and false positive rates were obtained. As just

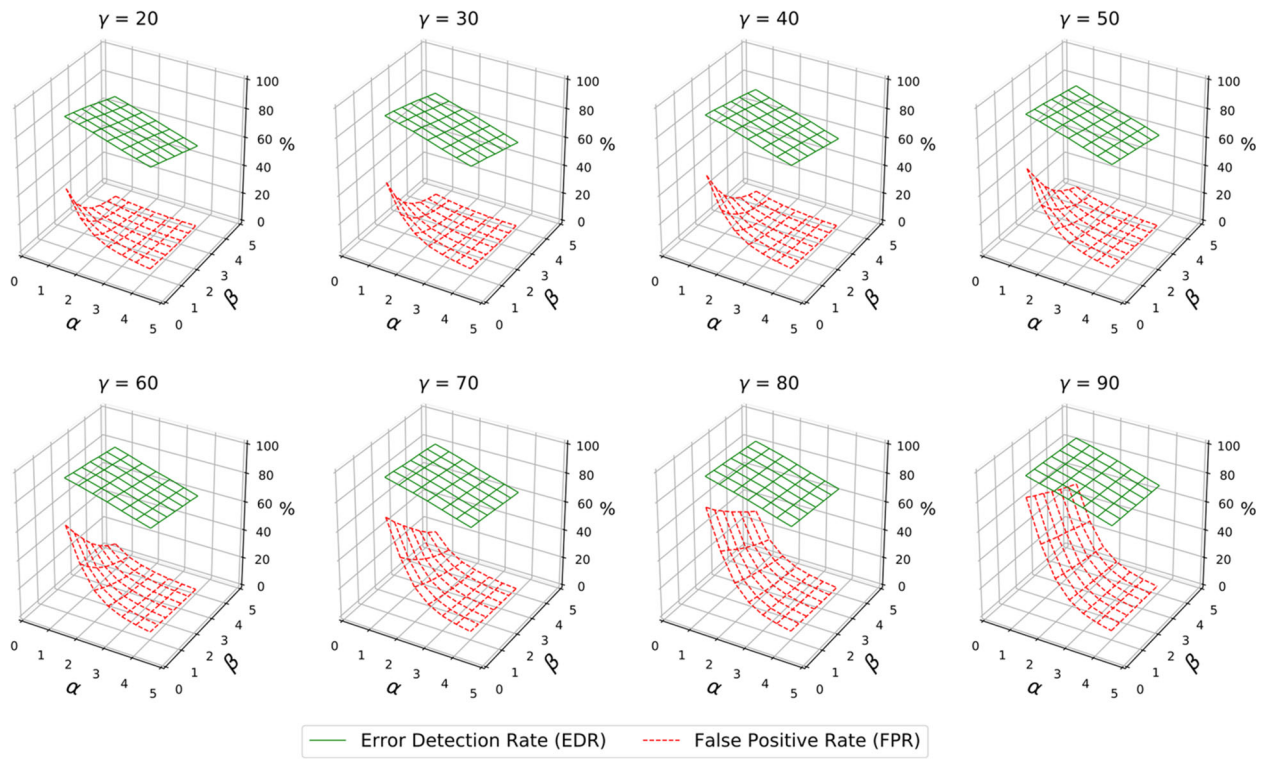


Fig. 5 α, β and γ influence on error detection rates and false positive rates—outlier errors

three numbers must be adjusted, thinking on a distributed environment as a WSN, it is necessary to broadcast a single message to adjust the algorithm's parameters on all nodes, or on a subset of nodes of the same type. As the MAE value is expected to be larger for models with highly varying values, the parameters can also be set individually for these type of nodes to get a better tolerance to the variance.

Error type coverage

A second set of evaluations was executed in order to identify the algorithm effectiveness on detecting the other three types of errors defined by [4]: *peak*, *stuck-at*, and *noise*. The error injections mechanism was adjusted to generate several sequences of errors in the same data chunks used in the first evaluation. Examples of the injected errors and their correspondent confidence levels are illustrated in Fig. 6. On these graphs, each type of errors was injected twice. To better identify each error type, they are marked on the graphs with different letters: **S** for *stuck-at* errors, **P** for *peak* errors, and **V** for *variance* (or *noise*) errors. The parameters used to create the graphs of Fig. 6 were $\alpha = 2.5$, $\beta = 2.5$, and $\gamma = 50$.

The peak errors are very similar to the outlier errors, used in the first evaluation. So, the confidence level drops as soon as the monitored values deviate enough from the usual (predicted) values. Figure 6 shows that peak errors are better detected than the stuck-at or variance errors. This happens because peak errors generate values farther from the expected ones. But in some situations, a peak can make the monitored values go into the directions of the predictions, mainly when there is some gap between the real values and the predicted ones. When this happens, the values remain in the range with confidence high enough, and the sequence may not be classified as an error. The second peak of the upper graph in Fig. 6 illustrates this, as the sensors readings get various confidence levels, from 0 to 80, when the peak error is occurring (around sample 300).

The detection of the *stuck-at* errors is determined by the data behavior. In the upper graph of Fig. 6, the repeated value is close to the predictions, making the error remain undetected. On the lower graph, however, as the predictions followed the input from the correlated variables, the difference from the constant value increased enough to make the errors be detected. This is reflected by the assignment of different levels of confidence, from 0 to 100.

The proposed mechanism gets low accuracy in identifying the *variance* error when the readings do not exceed the thresholds defined by the α and β parameters. The graphs of Fig. 6 shows that high variance errors (**V** marks) that keep the values close to the correct ones are not detected. Only when the noise produces values with larger discrepancies, the confidence of the variable starts to decrease. If the noise makes the monitored values fall too far from the expected ones, the proposed solution will detect them as it does with the outliers. The noise can be an indicator of malfunction or interference, but while it does not causes readings to be too distant from real values, the confidence attribution scheme will not classify this kind of behaviors as an error. This kind of behavior classification is not in the scope of this work. Solutions of variance analysis can be used at the server to indicate noise in a sequence of readings, indicating the need for some maintenance. Noise errors can also be detected if they produce readings outside the $\beta \times MAE$ threshold, producing sequences of different, highly varying levels of confidence. Again, this sensitiveness can be adjusted by setting the α and β parameters.

The "stuck-at" is an error type that can show a different impact on the monitored value confidence, depending on *when* it occurs and *how long* it lasts. When the monitored variable *"freezes"* on a value close to the normal values, and the last ones show only small variations, the error is not detected. This is a situation similar to the noise error and is illustrated in the first graph of Fig. 6. But when the normal values sequence is ascending or descending—like temperature in the morning or in the late afternoon—then the difference will be detected, and the monitored variable's confidence will fall in the same rate. This is well illustrated in the first error labeled **S** on the lower graph of Fig. 6. Obviously, if an error makes the readings to be stuck at a distant value, then the algorithm will drop the sensor confidence instantly.

The analysis of the results show that the algorithm can identify errors in sequences that are greater than some *threshold* and ignores errors that do not deviate too much from the *normal* values of the sequence. It does not detect *noise* and *stuck-at* error when they lay around the normal readings. This types of errors are easily detected by algorithms that use statistical analysis on a window of the N last readings. This type of detection and classification is out of the scope of this work. We are interested in identifying errors—from failures or intentionally injected by an intruder—that deviates from the correct values and could lead to wrong decisions, e.g., in a cyber-physical system, as well as labeling data with confidence levels to allow checking against message corruption when data arrives on the server.

The proposed algorithm labels every reading with an confidence level, adjustable by the parameters. So, even if not classified as an error by algorithm (falling bellow γ), every variation in the value of a sequence reflects in a variation in the confidence level assigned to it. A small (or even 0) value of β and a reasonable value of α make every different value be labeled with a different confidence level

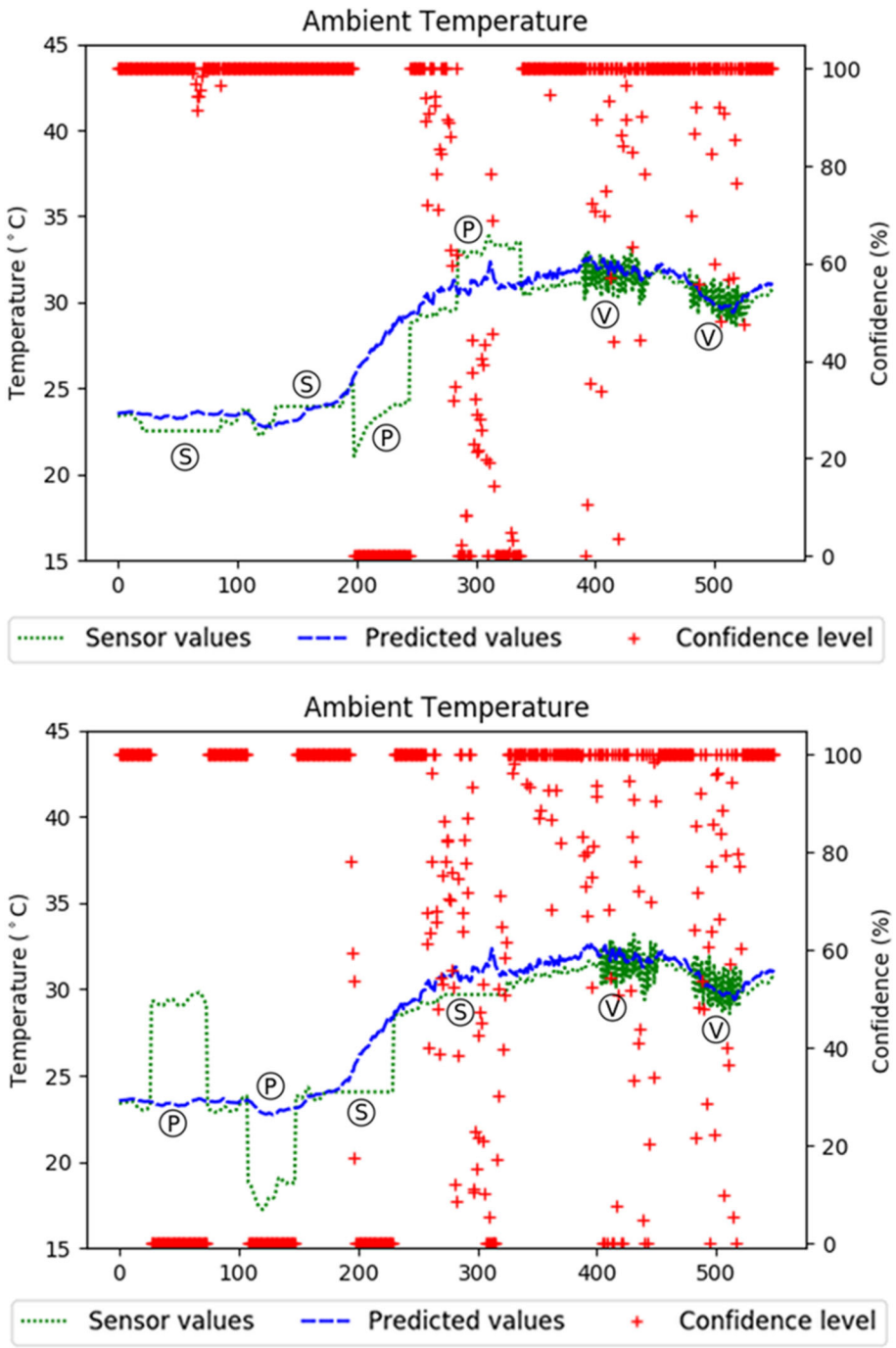


Fig. 6 Different error types injection

that can be used to identify *noise* and *stuck-at* errors at a second stage, not covered in this work.

To compare the results obtained with different error types, Fig. 7 shows the results for the combinations of the algorithm parameters α and β, with γ assuming 20, 50, and 90. Other values of γ were omitted due to lack of space. As already discussed, the EDR and FPR for the *stuck-at* and *variance* errors are significantly lower than for *outlier* (see Fig. 5) and *peak* errors. The peak errors also show an EDR slightly lower than the outlier errors, mainly by the fact that some peaks can produce values not far enough from the predicted values, and consequently not dropping the confidence enough to be detected as an error. As these peaks last much more time, they have

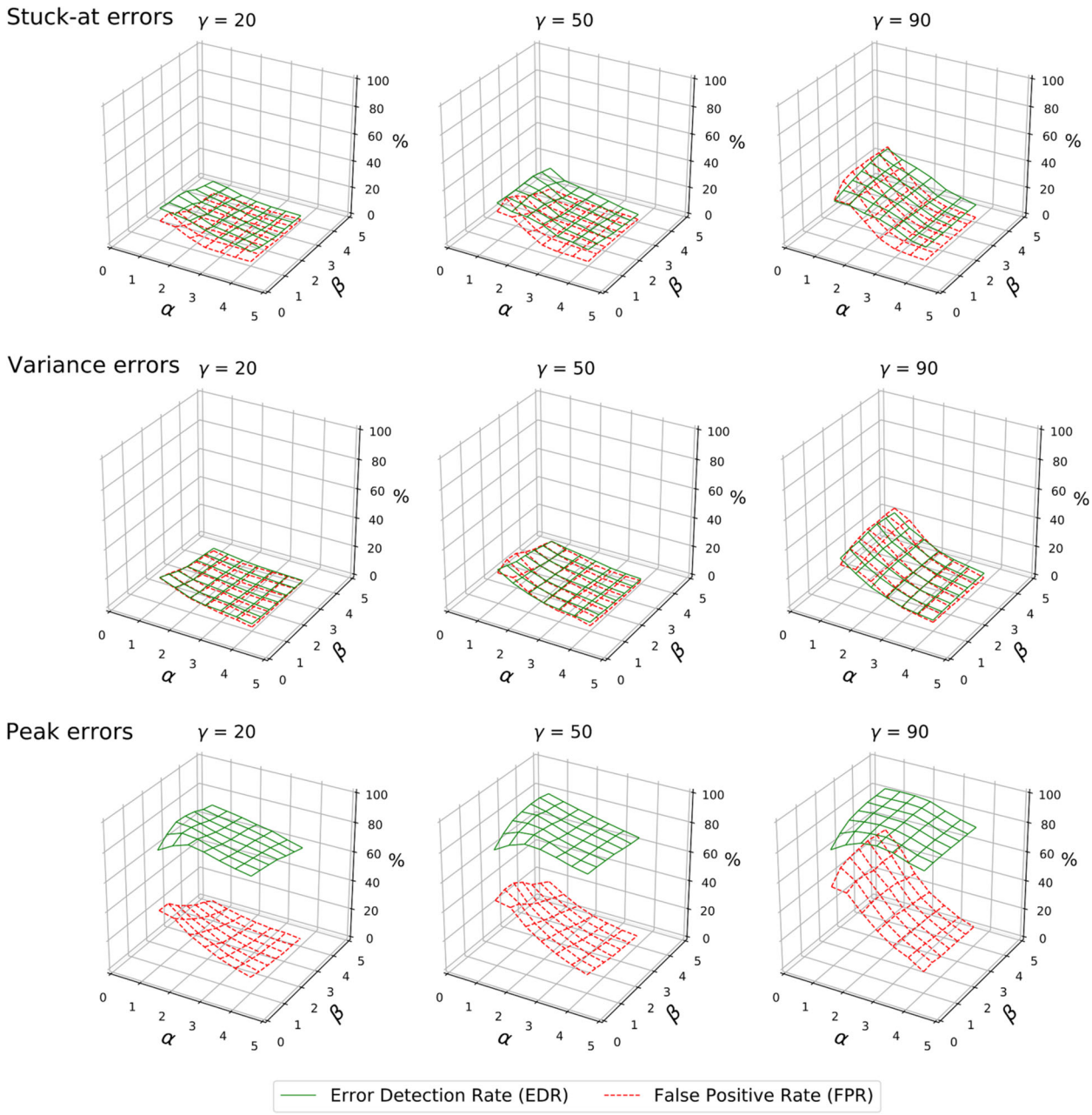


Fig. 7 Error detection rates and false positive rates for different types of errors

a more significant statistical impact that one outlier that does the same. If the simulation is adjusted to produce only large deviations when generating peak errors, the results would be practically the same to the obtained with the outlier errors. But this would artificially inflate the EDR and would not follow the original definition of the error type. The *stuck-at* and *noise* errors show much lower EDRs, as the error values were mainly too close to the predicted values. They may cause variation in the confidence assigned to the value while the error occurs, but are not reported as an error as the confidence remains above γ. This is the reason why $\gamma = 90$ associated with small values of α and β result in higher detection and false positive rates for both type of errors.

Comparison with the hybrid fault detection method

We compared our confidence attribution mechanism with the hybrid fault detection method proposed by [6], using data from the SensorScope project [30]. We applied our

mechanism to one of their datasets available online. The dataset consists of environmental data collected at the Grand Saint Bernard pass between Switzerland and Italy in 2007 [31]. It contains samples of temperature, humidity, and solar irradiation collected along 43 days with a temporal resolution of 2 min. The results obtained by applying our proposal to this dataset are compared to the results obtained by the authors [6]. They selected ten sensors among the most faulty ones to evaluate the accuracy of fault detection and classification and also applied time series analysis and neighbor voting algorithms to detect four types of failure.

We first aligned the datapoints in time. Two sensors that failed to produce data (their focus was on sensor failure, ours on data confidence) were excluded from the comparison. The resulting dataset contains 20180 datapoints. Subsequently, we manually classified discrepant datapoints as faulty. After that, sequences with no faulty data were selected at five different periods of the time series, resulting in a training set of 6800 records. The feature selection procedure was applied next, identifying the relationship among the variables across different sensors.

We built the ANNs as described in "Algorithm parameter evaluation" section and obtained the MAE for every sensor. The models were then evaluated for every sensor with average values: $\gamma = 50$, $\alpha = 2.5$, and $\beta = 2.5$. A node was considered faulty when the confidence reached 0. The compared work counted errors by occurrence in each test interval T of 30 minutes, no matter how long the error remained. As our proposed algorithm evaluates each datapoint individually, a direct comparison of error counts is meaningless. So, the error detection rate (which is called success ratio in their work) will be used for comparison. In their work, *neighbour voting (NV)* and *time series analysis (TS)* were combined to achieve the best results ($NV \cup TS$), which is the basis for the comparison with our mechanism summarized in Table 3.

The results show that for some sensors the EDR is significantly lower than the compared technique. Inspecting the data, it is possible to verify that the causes explained in "Error type coverage" section are present in the evaluated dataset. Taking as example sensor 18, it is visible that the errors are mainly of the *suck-at* type, as shown in Fig. 8. For clarity, confidence (red crosses) is only marked at 100 (no error) and 0 (error) to not hide data lines on the graph. The sensor reported −1 for a long time. As the real temperature was around that value for a significant period (around sample 3000), it was not detected as a faulty value because it was in the range of acceptable values. Almost all of the undetected errors are due to the fact that *stuck-at* errors are not identified when they are close to the real values. On sensors where mostly peak and outlier errors occur, as for sensor 7, 15, and 17, the error detection rate of our algorithm outperforms the compared solution. Figure 9 shows the detection of this type of errors on sensor 15. On the other hand, in [6] the authors demonstrate that the peak errors (named *drift* by the authors) is the one with lowest detection rates, reaching from 76.9 to 84.6% of accuracy.

Other solutions described on "Related work" section make their evaluations based on proprietary real or simulated WSN, so it was not possible to directly compare results, as data could not be replicated to be used with our algorithm.

Table 3 Compared algorithm evaluation

Sensor	NV ∪ TS (%)	Confidence attribution (%)
Sensor 6	97.4	75.1
Sensor 7	93.3	99.8
Sensor 9	94.1	90.7
Sensor 15	93.1	96.7
Sensor 17	91.8	93.4
Sensor 18	92.5	74.1
Sensor 19	92.5	83.9
Sensor 20	97.6	74.2

Concept drift detection

In this section, we discuss the detection of *concept drifts*. Once sensors may be monitoring dynamic environments, the simulation models are subject to modifications in the correlations between the readings of different sensors in different instants. As described by [32], a *concept* is the classification or prediction result of a vector of values α. If the result changes over time, i.e., $P_t(\chi) \neq P_u(\chi)$, then a *concept drift* occurs, as the same input set χ produces different results at times t and u, and both are correct. Similarly, [33] defines it as $\exists \chi : P_{t_0}(\chi, y) \neq P_{t_1}(\chi, y)$, meaning that the joint distribution of a set of input variables χ and the target variable y may vary from time t_0 to time t_1.

Concept drifts are not easily detected as they may be confused with long-lasting erroneous readings. Several methods for concept drift detection were proposed, mainly applying statistical analysis of errors and correlations. The first attempt in concept drift detection was the exponentially weighted moving average (EWMA) chart method proposed by [34], which verifies the changes in the moving average over the last N readings, with no need to keep buffers. However, when applied to a single time series, the algorithm classifies long error sequences as concept drifts, so it is mandatory to first classify if the readings are reliable, before applying the formulas. This makes it harder to apply this method directly into the proposed solution, as concept drifts may also be classified as errors, or at least as readings with lower confidence.

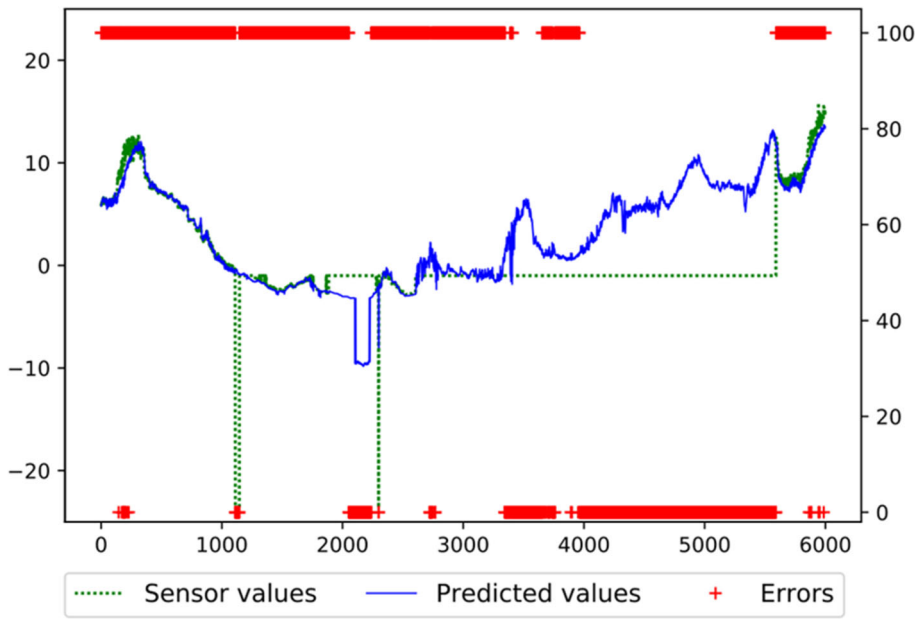


Fig. 8 Sensor 18 stuck at −1 for a long time

When a concept drift occurs, the predictors built at the server with outdated data will not be able to correctly handle that changes. The majority of the models will start to assign lower confidence levels to their data. For example, when the training is made with data obtained in summer days—as in the experiment presented in this work—the model assigns low confidence to relative humidity, ambient temperature, and barometric pressure, as they show correlations slightly different in winter (Fig. 10). As shown by the figure, the predicted values are clearly following the sensor readings but are not close enough to be considered correct.

At the server side—where enough computational power is available—more elaborated algorithms can be used to verify if some sequences are real concept drifts or long lasting errors, like sensors slowly deviating their readings from the correct ones. Models comparing readings variations of each sensor and comparing them to the variations in other sensors must be built, mixing statistical analysis and predictions results. These *ensembles*

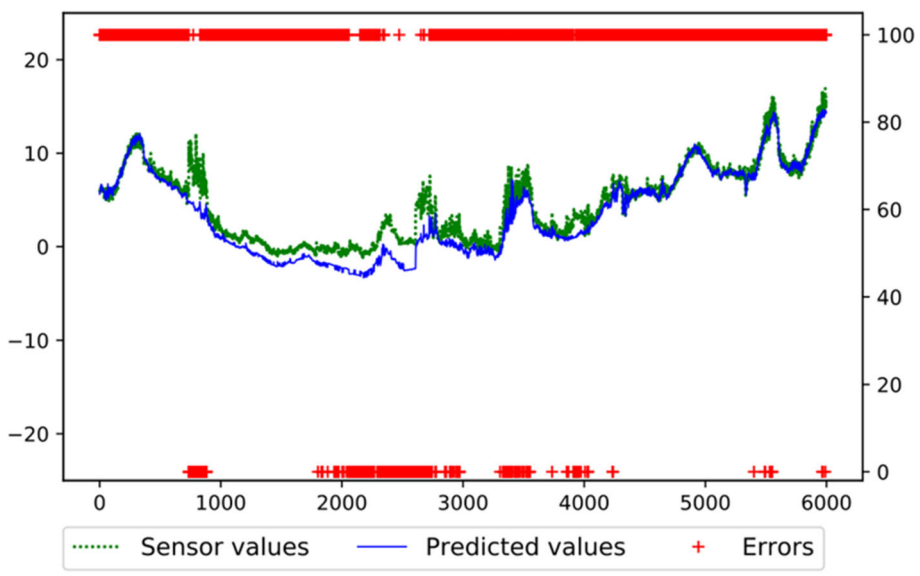


Fig. 9 Sensor 15 with peak and outlier errors

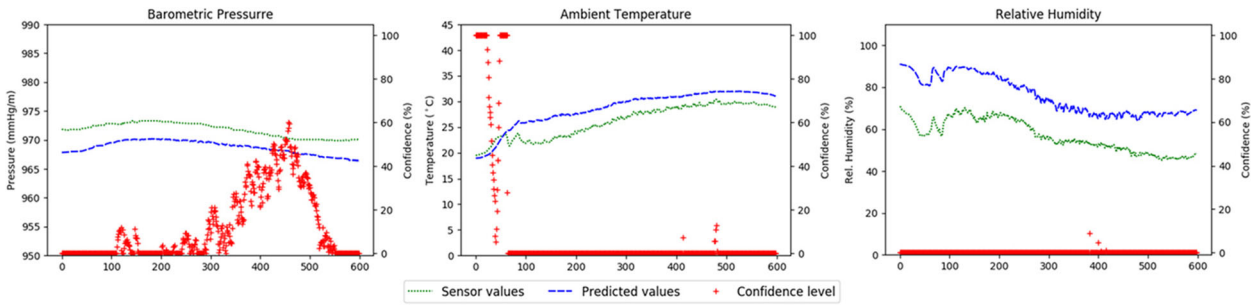


Fig. 10 General confidence drop on concept drift

can use some voting mechanism, for example, to decide if a concept drift has occurred. Another solution is use the outputs of all the different detection mechanisms and build and train a classifier to verify if the data streams should be considered errors or concept drifts.

We are evaluating the use of an algorithm that combines the outputs of a drift detection method such as the EWMA for the different available values. The idea is to evaluate the behavior of the values and the confidence levels of all correlated variables. The algorithm then has to decide if a set of deviations, occurring for a long enough period of time, is an error or a concept drift. We are also investigating other drift detection methods suitable for the given scenario.

Data confidence as trust mechanism

The proposed confidence attribution scheme has shown feasible to be applied to resource-constrained devices, as the model is not expensive in size to transmit and store in the nodes of the WSN. Neither demands high computational power to execute. The most power-demanding tasks—feature selection, model building, updating, and training—are performed offline, at powerful servers or on the Cloud.

As the models are built on the server, they can be stored to perform later verification to identify if data was adulterated at the origin (the sensor node), when it is retransmitted by intermediate nodes, or even by a malicious gateway. Therefore, it increases the overall dependability of the WSN data, as an attack have to occur in a coordinated way on all correlated nodes.

If an intruder forges the values read by a sensor ***before*** it enters the confidence attribution schema, it will be identified as faulty and get a low confidence level. The intruder can also try to change a sensor value ***after*** the confidence attribution. It can happen on the node, before the values are transmitted, on an relay node or at the gateway. In the last case, to be considered valid by the application and start an incorrect actuation, the false value has to be transmitted with high confidence attributed. Before sending the actuation command, the application can recalculate the confidence of the received value. It only has to apply the algorithm with correlated sensors values as input and check if the result is the same presented by the suspicious node. If a wrong value is presented with a high confidence value, it will not match the confidence calculated by the application. As another side effect, most confidence levels calculated by other nodes that used the false input will not match. The difference will be even worse if the neighbor nodes used the correct values in its calculations and the value is altered afterward, as at the gateway.

To forge plausible incorrect values, an intruder has to get access to all involved models and adjust the predicted values and the confidence levels of all correlated values that will be affected—directly or indirectly—by the forged value. It would be computationally expensive and inviable at the WSN level, due to the number of messages needed to adjust and verify all involved values. It could be possible at the gateway, but other intrusion detection techniques would easily detect the extra processing and retransmissions delays while the malicious software tries to calculate the new confidences.

Conclusions

Message-exchanging protocols for data fault detection are not suitable for wireless sensors networks due to resource constraints. A new scheme for data fault detection is presented in this work, based on local predictors and listening to the messages from correlated neighbors. The prediction models are built and updated outside the WSN and sent to the sensor nodes only when necessary, avoiding communication overheads. Nodes can determine a *confidence level* of the sensed value, comparing it with the predictor's output. The scheme can minimize the interference of a faulty value on other nodes, correctly identifying the defective node by assigning low confidence to the incorrect values. The algorithm's sensitiveness can be easily adjusted through the combination of three parameters (α, β, and γ) that can be broadcasted over the WSN with very low overhead.

Simulations shown that the proposed solution reaches good accuracy levels in identifying nodes' data faults and its source. A fault is a value too far from the expected correct value, based on other correlated observed values. Parameters can be adjusted to get the desired algorithm *sensitivity* or the balance between detection rate and false positives. The solution can also be used to verify the data integrity at the application, comparing models outputs with the received values at WSN gateways or at the application servers. As any interference on the read values and/or confidence levels can be compared to the output of other model, the proposed mechanism adds an extra authenticity and integrity level to the WSN.

As future work, efforts on the definition of a reliable change detection methods has to be made, in order to select methods that can address the several aspects involved in correctly identify the occurrence of *concept drifts*. The results of our preliminary results shown that the statistical and time series analysis on single data streams have to be reinforced by comparing data from different sources. It is expected that such *ensembles* can reach better results in this task. On the predictors, the use of Incremental Gaussian Mixture Network (IGMN) [35] will also be evaluated, as that model is resilient to missing inputs. This feature can be useful in networks subject to communication problems, in unreliable environments. The integration of the proposed solution to a communication protocol will also be implemented and evaluated, in order to measure the real impact on energy consumption and the detection of possibly intrusion attempts in WSN and CPS scenarios.

Acknowledgements
The authors would like to thank the support provided by the involved universities.

Authors' contributions
Both authors contributed equally to this work. Both authors read and approved the final manuscript.

Funding
Not applicable.

Availability of supporting data
The dataset used in the experiments is avaliable at https://lisha.ufsc.br/tiki-list_file_gallery.php?galleryId=52.

Competing interests
The authors declare that they have no competing interests.

Received: 5 April 2019 Accepted: 4 November 2019
Published online: 12 December 2019

References

1. Scheffel RM, Fröhlich AA (2018) WSN data confidence attribution using predictors. In: 2018 Eighth Latin-American Symposium on Dependable Computing (LADC). IEEE, Piscataway. pp 145–154
2. Khan MZ (2013) Fault management in wireless sensor networks. Comput Sci Telecommun 37(1):3–17
3. Ni K, Ramanathan N, Chehade MNH, Balzano L, Nair S, Zahedi S, Kohler E, Pottie G, Hansen M, Srivastava M (2009) Sensor network data fault types. ACM Trans Sensor Netw (TOSN) 5(3):25
4. Sharma AB, Golubchik L, Govindan R (2010) Sensor faults: detection methods and prevalence in real-world datasets. ACM Trans Sensor Netw (TOSN) 6(3):23
5. Moustapha AI, Selmic RR (2008) Wireless sensor network modeling using modified recurrent neural networks: application to fault detection. IEEE Trans Instrument Measure 57(5):981–988
6. Nguyen TA, Bucur D, Aiello M, Tei K (2013) Applying time series analysis and neighbourhood voting in a decentralised approach for fault detection and classification in WSNs. In: Proceedings of the Fourth Symposium on Information and Communication Technology. ACM, New York. pp 234–241
7. Li W, Bassi F, Dardari D, Kieffer M, Pasolini G (2015) Low-complexity distributed fault detection for wireless sensor networks. In: Communications (ICC), 2015 IEEE International Conference On. IEEE, Piscataway. pp 6712–6718
8. Panda M, Khilar PM (2012) Distributed soft fault detection algorithm in wireless sensor networks using statistical test. In: Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference On. IEEE, Piscataway. pp 195–198
9. Panda M, Khilar PM (2015) Distributed self fault diagnosis algorithm for large scale wireless sensor networks using modified three sigma edit test. Ad Hoc Netw 25:170–184
10. Yuan H, Zhao X, Yu L (2015) A distributed Bayesian algorithm for data fault detection in wireless sensor networks. In: Information Networking (ICOIN), 2015 International Conference On. IEEE, Piscataway. pp 63–68
11. Saihi M, Boussaid B, Zouinkhi A, Abdelkrim N (2015) Distributed fault detection based on hmm for wireless sensor networks. In: Systems and Control (ICSC), 2015 4th International Conference On. IEEE, Piscataway. pp 189–193
12. Liu K, Zhuang Y, Wang Z, Ma J (2015) Spatiotemporal correlation based fault-tolerant event detection in wireless sensor networks. Intl J Distrib Sensor Netw 11(10):643570
13. Titouna C, Aliouat M, Gueroui M (2016) FDS: fault detection scheme for wireless sensor networks. Wirel Person Commun 86(2):549–562
14. Titouna C, Gueroui M, Aliouat M, Ari AAA, Amine A (2017) Distributed fault-tolerant algorithm for wireless sensor network. Int J Commun Netw Inf Secur 9(2):241
15. Zhang T, Zhao Q, Nakamoto Y (2017) Faulty sensor data detection in wireless sensor networks using logistical regression. In: Distributed Computing Systems Workshops (ICDCSW), 2017 IEEE 37th International Conference On. IEEE, Piscataway. pp 13–18
16. Shao S, Guo S, Qiu X (2017) Distributed fault detection based on credibility and cooperation for WSNs in smart grids. Sensors 17(5):983
17. Swain RR, Khilar PM (2017) Composite fault diagnosis in wireless sensor networks using neural networks. Wirel Person Commun 95(3):2507–2548
18. Woźniak M, Graña M, Corchado E (2014) A survey of multiple classifier systems as hybrid systems. Inf Fusion 16:3–17
19. Curiac D-I, Volosencu C (2012) Ensemble based sensing anomaly detection in wireless sensor networks. Expert Syst Appl 39(10):9087–9096
20. O'Reilly C, Gluhak A, Imran MA, Rajasegarar S (2014) Anomaly detection in wireless sensor networks in a non-stationary environment. IEEE Commun Surveys Tutor 16(3):1413–1432
21. Ditzler G, Roveri M, Alippi C, Polikar R (2015) Learning in nonstationary environments: a survey. IEEE Comput Intell Mag 10(4):12–25
22. Visalakshi S, Radha V (2014) A literature review of feature selection techniques and applications: review of feature selection in data mining. In: Computational Intelligence and Computing Research (ICCIC), 2014 IEEE International Conference On. IEEE, Piscataway. pp 1–6
23. Frank E, Hall MA, Witten IH (2017) Data Mining - Practical Machine Learning Tools and Techniques. 4th ed.. Morgan Kaufmann, Cambridge. pp. 553–571
24. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, et al (2011) Scikit-learn: machine learning in python. J Mach Learn Res 12(Oct):2825–2830
25. Trenn S (2008) Multilayer perceptrons: Approximation order and necessary number of hidden units. IEEE Trans Neural Netw 19(5):836–844
26. Thomas P, Suhner M-C (2015) A new multilayer perceptron pruning algorithm for classification and regression applications. Neural Process Lett 42(2):437–458
27. Willmott CJ, Matsuura K (2005) Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. Climate Res 30(1):79–82

28. Al-Rfou R, Alain G, Almahairi A, Angermueller C, Bahdanau D, Ballas N, Bastien F, Bayer J, Belikov A, Belopolsky A, et al (2016) Theano: a python framework for fast computation of mathematical expressions. arXiv preprint arXiv:1605.02688 472:473
29. Chollet F (2015) Keras: The Python Deep Learning library. https://keras.io. Accessed 27 Aug 2018
30. Barrenetxea G, Ingelrest F, Schaefer G, Vetterli M, Couach O, Parlange M (2008) Sensorscope: out-of-the-box environmental monitoring. In: Proceedings of the 7th International Conference on Information Processing in Sensor Networks. IEEE Computer Society, Piscataway. pp 332–343
31. Sensor Scope Dataset. http://sensorscope.epfl.ch/. Accessed 26 Feb 2019
32. Webb GI, Hyde R, Cao H, Nguyen HL, Petitjean F (2016) Characterizing concept drift. Data Mining Knowl Discovery 30(4):964–994. https://doi.org/10.1007/s10618-015-0448-4
33. Gama J, Žliobaitė I, Bifet A, Pechenizkiy M, Bouchachia A (2014) A survey on concept drift adaptation. ACM Comput Surv (CSUR) 46(4):44
34. Ross GJ, Adams NM, Tasoulis DK, Hand DJ (2012) Exponentially weighted moving average charts for detecting concept drift. Pattern Recogn Lett 33(2):191–198
35. Heinen MR, Engel PM, Pinto RC (2011) Igmn: an incremental Gaussian mixture network that learns instantaneously from data flows. Proc VIII Encontro Nacional de Inteligência Artificial (ENIA2011):488–499

Publisher's Note