A Decentralized Scheme for Fault Detection and Classification in WSNs

Tuan Anh Nguyen and Marco Aiello
University of Groningen
Groningen, The Netherlands
Email: {t.a.nguyen, m.aiello}@rug.nl

Kenji Tei
National Institute of Informatics
Tokyo, Japan
Email: tei@nii.ac.jp

***Abstract*—In pervasive computing environments, a context-aware application captures the current real-world context and accordingly controls various embedded devices to achieve the desired conditions, supporting its users. Context information is made available and shareable for the application by using wireless sensor networks (WSNs) thanks to the provision of sensors and actuators wirelessly interconnected. WSNs are deployed to monitor the phenomena of interests and the reporting should be as accurate and reliable as possible. However sensor nodes are usually battery-powered, also they have small memory and computational capacity as well as limited communication bandwidth. Such tight resource constraints often hinders the reliability of the monitoring. Thus sensor data usually suffers from both discontinuous (random or malfunction) and continuous (bias or drift) faults. Therefore fault detection and classification is obvious to ensure the reliability of context information provided. In this paper we propose a decentralized scheme not only for fault detection but also for their classification. In our solution, neighbourhood voting and the ARMA time series forecasting model are used in combination to detect sensor data faults. Then, the detected faulty readings are classified based on the frequency and continuity of fault occurrence and the observable and learnable patterns that faults leave in the data. Our solution also uses histogram analysis to improve the overall accuracy. The proposed solution is the first step of realizing a fault handling mechanism on sensor nodes so that each node can handle its own faulty readings locally in a real-time manner instead of depending on reactions from a base station. Such a decentralized scheme is in favour to satisfy the requirements of online sensor data fault tolerance (e.g., scalability and real-timeness).**

I. Introduction

In pervasive computing environments, a context-aware application captures the current real-world context and accordingly controls various embedded devices to achieve the desired conditions, supporting its users. Context information is made available and shareable for the application by using a context management system (CMS). The CMS should be able to capture various kinds of context required by context-aware applications and should be able to be deployed easily to various environments [10]. Thanks to the provision of sensors, actuators, and wireless communication capability, WSNs become the ideal infrastructure for the context management system.

WSNs monitor the phenomena of interest and are responsible for producing accurate and reliable data about monitored environments. However, the sensor nodes are deployed in harsh environments and left unattended after the calibration phase, what is more, that is the limited capacity and energy source of wireless sensor nodes. In such conditions, it is inevitable that some sensor nodes malfunction, thus WSNs become unreliable and context information becomes faulty. Therefore, fault detection and classification are obvious to ensure the reliability and accuracy of context information provided.

Much research has been focusing on fault tolerance in WSNs. The fault tolerance mechanisms are generally divided into two main strands. The first strand of proposed fault tolerance mechanisms in WSNs is centralized, which takes advantage of computational capacity of a base station for data analysis. The disadvantages of this strand lay in its limited scalability in real-time handling distributed sensor data streaming as well as in necessitating high amounts of communication with the nodes. The second strand considers decentralization as its scheme, in which nodes flexibly handle their own faulty readings locally, in a faster manner, without waiting for reactions from a base station. Therefore, a decentralized scheme is in favour to satisfy the requirements of online sensor fault correction, such as scalability and real-timeness, maintaining the comparable level of accuracy. When designing a decentralized technique, the challenge is to process as much data as possible while keeping the communication overhead, memory and computational cost low [11].

Motivated by the requirements considered by online sensory data fault correction, we propose a decentralized scheme not only for fault detection but also for their classification. The basis of our fault handling research is the possibility to detect and diagnose faults in sensory readings through the trace they leave in the data behaviour. This trace forms a visible pattern that is independent of the underlying cause of the fault [3]. Therefore, we can define a fault as a manifestation of erroneous readings. We focus on detecting and classifying faults that occur in sensor readings within data-centric WSNs. In this type of sensor networks, readings from sensors are spatially and temporally correlated and the development of faults in different nodes are uncorrelated.

At the detection phase, our proposed solution is based on a hybrid approach using a neighbourhood vote technique and an AutoRegressive–Moving-Average (ARMA) model for time series data forecasting. Faulty state of a reading is identified by comparing the reading's value with 1) the value computed by neighbourhood voting, and 2) the value predicted with the ARMA time series forecasting model. Decision on faulty state can be based on either an intersection or an union of the two comparisons. Neighbourhood voting technique is adopted as it does not require a priori knowledge of the phenomenon. Instead, this technique takes the advantage of

redundancy in measurements of sensor readings. The ARMA model is chosen because it has been shown to be theoretically and experimentally a strong candidate for time series predictions [4]. Moreover, the model parameters can be adapted to the underlying time series in an online fashion, without the need of storing large sets of past data [6].

At the phase of classification, in order to classify faulty readings we refer to the frequency and continuity of fault occurrence and the observable and learnable patterns that faults leave in the data. Histogram analysis is aslo used in order to distinguish between bias and drift faults. Our solution is able to detect and classify faulty sensor readings into four types of faults, namely, 1) random, 2) malfunction, 3) bias, and 4) drift. The proposed solution is the first step of realizing a decentralised fault handling mechanism on each node. Such a decentralized scheme is in favour to satisfy the requirements of online sensor data fault tolerance (e.g., scalability and real-timeness).

II. Sensor Data Fault Modelling

There is no doubt that faults appear often in sensor data. The causes of faulty readings could be the tight resource constraints of sensor nodes, such as battery-powered, small memory and computational capacity as well as limited communication bandwidth. As the first step of fault handling process, it is crucial to categorize faults into different sets. By comprehending the causes, effect, and especially the characteristics of each fault type, it is possible to propose appropriate fault tolerance mechanisms to detect, classify, and correct faults of each type.

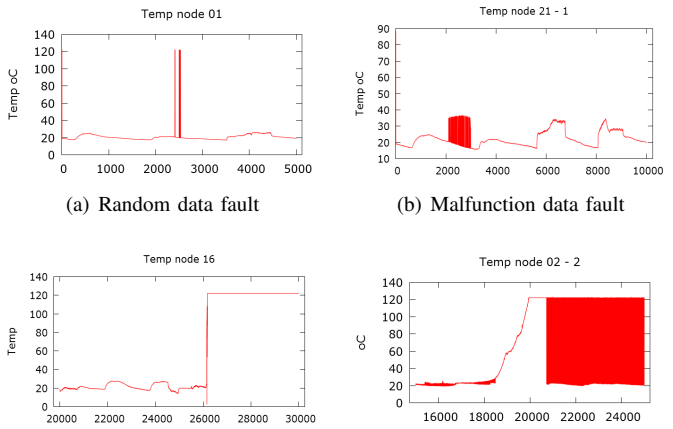
Fault categorization may vary depending on the points of view. Literature exists several fault taxonomies [7], [9] formed using different criteria, such as cause, impact, or duration of faults. One can also categorize faults based on the layer of the network stack where the fault occurs. For example, random noise, malfunctioning or, probably most common, calibration systematic error may appear at the physical layer. In terms of duration, faults can be classified as permanent, intermittent or transient. Ni et al. [7] give extensive taxonomies of faults that cover definitions, causes, duration and impact of faults. According to the categorization of the authors, sensor network faults can be classified into two broad fault types, namely, 1) *system faults* and 2) *data faults*. From system centric view point, faults may caused by *calibration*, *low battery*, *clipping*, or *environment out of range*. On the other hand, data faults comprise *stuck-at*, *offset*, and *gain* faults. These three types of data faults also are named as *short*, *constant*, and *noise*, respectively, by Sharma et al. in [9].

Considering the definitions of faults from a different view point, Baljak et al. [2] propose a complete and consistent categorization based on **the frequency and continuity of occurrence** and **observable and learnable pattern** that faults leave in the data. Their approach shares the same point of view with ours, thus we use these fault taxonomies in our research. We believe that the categorization is flexible and applicable to a wide range of sensor readings. The underlying cause of the error does not affect this categorization, thus it possible to handle the faults based on their pattern occurrences on each sensor node. In our model, there are four types of sensor data faults as follows,

- **Discontinuous** – Faults occur from time to time, the occurrence of faults is discrete.
  - **Malfunction** – Faulty readings appear frequently. The frequency of the occurrences of faults is higher than a threshold τ.
  - **Random** – Faults just appear randomly. The frequency of the occurrences of faulty readings smaller than τ.
- **Continuous** – During the period of checking, a sensor returns constantly inaccurate readings, and it is possible to observe a pattern in the form of a function.
  - **Bias** – The function of the error is a constant. This can be a positive or a negative offset.
  - **Drift** – The deviation of data follows a learn able function, such as a polynomial change.

Examples of sensor data fault types of temperature readings form Intel Lab at Berkeley dataset[1] are illustrated in Figure 1. More detailed explanations of our fault modelling are presented in [3].

By comprehending the type of faults, each node is able to handle its faulty readings appropriately, according to the type of fault. Random faults can be discarded as they does not contain any meaningful information. Malfunction nodes should be removed from the network. Bias and drift faults are more interesting because they can be corrected if a suitable fault model learning is applied.



(a) Random data fault (b) Malfunction data fault

(c) Bias data fault (d) Drift data fault then malfunction

Fig. 1. Examples of data fault types in temperature sensor readings from Intel Lab at Berkeley dataset.

III. Distributed Fault Handling

Practically, faulty readings are detected and classified at once, however, to make it more understandable and explainable we consider the detection and classification as two separate phases. Our solution uses a combination of neighbourhood voting, time series data analysis, and histogram analysis techniques for detecting faulty readings. Details of this hybrid approach for fault detection are explained in Section III-A below. Regarding fault classification, we take into account the duration and continuity that faults produce in sensor readings. In other

[1] http://db.csail.mit.edu/labdata/labdata.html

words, we focus on how long and how often faults appear in the checking period. The classification algorithm is realized based on the fault model that we proposed and discussed in Section II. In the following, we describe in more details our hybrid solution that can be realized on each sensor node, providing nodes the ability to not only check the faultiness of its readings, but also to classify its faults.

A. Hybrid Fault Detection

At the detection phase, a sensor node compares its current reading's value with 1) the value computed by neighbourhood voting, and 2) the expected value previously forecast with the ARMA time series data forecasting model. The faulty state of the reading is decided based on either an intersection or an union of the two comparisons. The former technique would decrease the fault positive while the latter one decreases the fault negative. The result of the detection phase is the faulty state of the sensor readings.

1) Neighbourhood Voting: The basis of our neighbourhood voting technique is that 1) neighbourhood nodes are deployed closely together, thus the distance between them is one single-hop transmission, 2) the nodes monitor the same phenomenon, and 3) faults at each node develops uncorrelated. In a nutshell, neighbourhood voting technique expects that the majority of the sensors report the correct value of the monitored phenomenon, thus a specific node can relay on its neighbours to check the validity of its reading. With neighbourhood voting technique, a priori knowledge of the phenomenon is not necessary. Instead, it takes the advantage of redundancy in measurements of sensor readings.

In our approach, we define two sensor nodes are neighbours if they can communicate with each other by using single-hop transmission. Given a node S_i, its reading r_i, the set of its neighbours is noted by $Neighbour(S_i)$. The number of its neighbours is denoted as $|Neighbour(S_i)|$. The neighbourhood voting technique used at each sensor node S_i includes following steps:

1) Collect the group of readings $R = r[1..|Neighbour(S_i)|]$ from all of its neighbours, excluding its own reading r_i.
2) Calculate the median of the group, $\mu = \{R\}_{\frac{1}{2}} = \tilde{r}$.
3) Calculate the difference between r_i and $\tilde{r}$,
$$D_{r_i\tilde{r}} = |r_i - \tilde{r}|.$$
4) Compare the difference $D_{r_i\tilde{r}}$ with a threshold τ, that can be adjusted but usually set at $\tau = 0.2 * \tilde{r}$
   - If $D_{r_i\tilde{r}} < \tau$ then r_i is a good reading,
   - If $D_{r_i\tilde{r}} \geq \tau$ then r_i is a faulty reading.

2) Time Series Analysis: Sensor measurements are the observations of a well-defined phenomenon monitored. The observations are obtained periodically over time. In addition, the measurements exhibit temporal correlation between consecutive observations. Thus sensor data are good representative of time series data. Taking this advantage, time series data analysis could be used for fault detection. The idea is to use a suitable time series forecasting model to predict the sensor reading at time t based on $t - k$ previous readings known in advance. When actual observation at time t is available, it is compared against its predicted value to determine if it is faulty. In this paper, we choose the ARMA model for two reasons. First, the model has been shown to be both theoretically and experimentally good candidates for time series predictions [4]. Second, model parameters can be estimated using the recursive least square (RLS) algorithm [1], which allows to adapt the parameters to the underlying time series in an online fashion, without the need of storing large sets of past data.

*a) **Time series data**:* A Time series (TS) is a collection of observations made sequentially, usually in time [4]. We denote a TS as a variable of X as $X = (X_1, X_2, ..., X_t, ...)$ or $X = \{X_t\}_{t=1,2,...}$. Given a time series of data $\{X_t\}$, the future values of this time series data can be predicted by using a suitable time serires forecasting model. In our proposal, we investing the use of the ARMA model for sensor time series data.

*b) **AutoRegressive–Moving Average model**:* The notation $ARMA(p, q)$ refers to the model with p autoregressive terms and q moving-average terms. This model contains the AutoRegressive $AR(p)$ and Moving-Average $MA(q)$ models. The $ARMA(p, q)$ model is written as

$$\begin{aligned} X_t = \phi_1 X_{t-1} + \phi_2 Xt - 2 + ... + \phi_p X_{t-p} \\ + a_t - \theta_1 a_{t-1} - \theta_2 at - 2 - ... - \theta_q a_{t-q} \end{aligned} \quad (1)$$

where we use the symbols $\phi_1, \phi_2, ..., \phi_p$ and $-\theta_1, -\theta_2, ..., -\theta_q$ are two finite sets of parameters for $AR(q)$ and $MA(p)$, respectively, and a_t is white noise.

In practice, the autoregressive and the moving average models of first order $(p = 1)$, $(q = 1)$, and of second order $(p = 2)$, $(q = 2)$ are of considerable practical importance [4]. Thus in our implementation, we use the $ARMA(2, 2)$ model of second order $(p = 2, q = 2)$.

*c) **Parameter Estimation**:* We used the exact maximum likelihood (ML) computational method [5] to estimate the two set parameters, $\phi_1, \phi_2, ..., \phi_p$ and $-\theta_1, -\theta_2, ..., -\theta_q$, of the model using training data.

The parameter estimation phase is performed at a base station by using traning data from each sensor. After that, the ARMA model with determined parameters is implemented on sensor nodes. Practically, estimating parameters before actual deployment of the sensors is viable because sensor nodes usually are and should be well calibrated in advance. At the running time, the base station that collects data from all sensors can re-perform the parameter estimation periodically for each sensor and sends the updated parameters to the proper nodes. By this way, the ARMA model at good sensor nodes can adapt with the change in the phenomenon monitored.

*d) **Fault detection**:* To detect faults in a sensor measurement time series, a sensor node first forecasts its reading at time t by using the *ARMA* model. When actual reading is available, sensor node computes the difference between actual sensor measurement at time t and its predicted value, and flags the measurement as faulty if this difference is above a threshold δ that is also known as *confidence interval* and usually set at 95%.

B. Fault Classification Algorithm

Once a sensor node exhibits faulty measurements, the classification process checks the frequency and continuity of the

occurrence of faults in order to identify the fault type, i.e., either random, malfunction, bias, drift, that the node is suffering. One should notice that while sensor reading is checked for its faulty state immediately, i.e., right at the time when the reading is observed, together with the readings from its neighbour, the classification process runs periodically, i.e., after every T readings in order to be able to check the frequency and continuity of the occurrence of faults. The number of readings T, thus the time interval for fault classification, is adjustable to meet the requirement of real-timeness. The classification process is clarified in Algorithm 1. The algorithm takes, as

Algorithm 1 Fault Classification

Input: 1) $R[1..T]$: vector of T sensor readings
2) $S[1..T]$: vector of the faulty state of $R[1..T]$
Output: C: fault type of sensor node in the interval
1: Compute the occurrences of faults $|\varepsilon_i|$— in R
2: check the continuity
3: **if** ε_i is discrete **then**
4: Check the frequency
5: **if** $|\varepsilon_i| \geq \tau$ **then**
6: $C = Malfunction$
7: **else**
8: $C = Random$
9: **end if**
10: **end if**
11: **if** ε_i is continuous **then**
12: Check the fault function ε_i
13: **if** $\varepsilon_i = const$ **then**
14: $C = Bias$
15: **else**
16: $C = Drift$
17: **end if**
18: **end if**
19: return C

its inputs, the $R[1..T]$ readings within the checked interval together with the faulty state (i.e., good or faulty) of the readings. First, the occurrences of faulty readings $|\varepsilon_i|$ in R are computed. Next, the algorithm checks the continuity and the frequency of the occurrences of faults $|\varepsilon_i|$ in order to classify the faulty state of the sensor node in the current interval into one of four fault types (random, malfunction, bias, drift).

*e) **Histogram analysis for checking the constant of** ε_i:* The histogram represents the frequency of occurrence by classes of data [8]. In our case, the histogram shows how faulty readings distribute. In order to check if the fault type is bias, we check the histogram of the readings $R[1..T]$. If the frequency of occurrence of less than c classes of data is higher than τ, we say that the fault is bias, otherwise the fault type is drift. In practice, c usually is 2 or 3, while τ is assigned by 0.8.

IV. Future Work

Faults exhibit frequently in sensor data. Faulty readings can be corrected with fault model learning if faults are detected and classified properly. By this way, context information provided by WSNs is more reliable and accurate. As the first steps of designing an online fault handling framework, in this paper we propose a decentralized scheme for fault detection and classification. Our intention is to realize a fault handling framework on each sensor node so that each node can handle its own faulty readings locally in real-time manner instead of depending on reactions from a base station. Such a decentralized framework is in favour to satisfy the requirements of online sensor data fault tolerance (e.g., scalability and real-timeness).

Our proposed solution applies both neighbourhood voting mechanism and time series data analysis in combination to detect sensor data faults. Sensor reading is compared with not just the value computed by neighbourhood voting, but also is double checked with the value previously forecast by the ARMA time series data model. Then, the detected faulty readings are classified based on the frequency and continuity of fault occurrence and the observable and learnable patterns that faults leave in the data. In addition, histogram analysis checks the constant of faults, distinguishing bias and drift faults. In our solution, the ARMA time series forecasting modelling is chosen as it shows the potential to be a lightweight mechanism, satisfying the requirements of keeping the communication overhead, memory and computational cost low. Thus we plan to implement our hybrid solution and realize it on actual sensor nodes in order to evaluation our solution in terms of these important requirements.

References

[1] S. T. Alexander. *Adaptive Signal Processing: Theory and Applications*. Springer-Verlag New York, Inc., 1986.

[2] V. Baljak, T. Kenji, and S. Honiden. Faults in Sensory Readings: Classification and Model Learning. *Sensors & Transducers*, 18:177–187, 2013.

[3] V. Baljak, K. Tei, and S. Honiden. Classification of faults in sensor readings with statistical pattern recognition. In *SENSORCOMM 2012, The Sixth International Conference on Sensor Technologies and Applications*, pages 270–276, 2012.

[4] G. E. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis: Forecasting and Control*. Wiley.com, 2013.

[5] Y. Bresler and A. Macovski. Exact maximum likelihood parameter estimation of superimposed exponential signals in noise. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 34(5):1081–1089, 1986.

[6] Y.-A. Le Borgne, S. Santini, and G. Bontempi. Adaptive Model Selection for Time Series Prediction in Wireless Sensor Networks. *Signal Processing*, 87(12):3010–3020, 2007.

[7] K. Ni, N. Ramanathan, M. N. H. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen, and M. Srivastava. Sensor Network Data Fault Types. *ACM Transactions on Sensor Networks (TOSN)*, 5(3):25, 2009.

[8] K. Pearson. Contributions to the Mathematical Theory of Evolution. II. Skew Variation in Homogeneous Material. *Philosophical Transactions of the Royal Society of London. A*, 186:343–414, 1895.

[9] A. B. Sharma, L. Golubchik, and R. Govindan. Sensor Faults: Detection Methods and Prevalence in Real-World Datasets. *ACM Transactions on Sensor Networks (TOSN)*, 6(3):23, 2010.

[10] K. Tei, S. Suenaga, Y. Nakamura, Y. Sei, H. Nakazato, Y. Kaneki, N. Yoshioka, Y. Fukazawa, and S. Honiden. XAC Project: Towards a Middleware for Open Wireless Sensor Networks. *Designing Solutions-Based Ubiquitous and Pervasive Computing: New Issues and Trends*, 1:214–231, March 2010.

[11] Y. Zhang, N. Meratnia, and P. Havinga. Outlier Detection Techniques for Wireless Sensor Networks: A Survey. *Communications Surveys & Tutorials, IEEE*, 12(2):159–170, 2010.