Data collection with in-network fault detection based on spatial correlation

Lei Fang, Simon Dobson
School of Computer Science
University of St Andrews, UK
Email: lf28@st-andrews.ac.uk, simon.dobson@st-andrews.ac.uk

March 3, 2015

Abstract

Environmental sensing exposes sensor nodes to environmental stresses that can lead to various kinds of sampling failure. Recognising such faults in the network can improve data reliability therefore making sensor networks suitable candidate for critical monitoring applications. We develop a technique that builds a spatial model of a sensor network and its observations, and show how this can be updated in-network to provide outlier detection even for non-stationary time series. The solution does not require local storage of learning data or any centralised control. The method is evaluated by both real world implementation and simulation, and the results are promising.

Key words: Fault detection, Sensor networks, Online learning, Energy efficiency

1 Introduction

Wireless sensor networks (WSNs) are prime candidates for the application of both autonomic and cloud computing techniques. Considered in the large, they observe, collect, and deliver events and/or data from a distributed system to a data store for further analysis. On a small scale, they must function in a hostile environment with limited power budgets, communications and computational capabilities. Since the data observed by WSNs may later be used for scientific or policy purposes, it is vital that the results can be trusted, despite the lack of human-in-the-loop oversight and independent ground truth.

The performance of individual nodes in a WSN may vary for a variety of reasons, including hardware faults, direct or indirect environmental action, or deliberate attack. While the observations of a single node are questionable, the existence of observations from other, nearby nodes observing the same phenomena may be used to provide corroborating evidence, allowing the network as a whole to be more accurate than its individual components. The situation is however complicated by two factors. First, the phenomena being observed are by definition dynamic, and may present discontinuous rapid changes between more stable regimes (such as when temperature changes rapidly before a storm). Failing at recognising this change will lead to high false alarm rates, as good data will be classified as faults for its mismatch with the stale model. Second, as resource constrained computing devices, sensors cannot host computational expensive models nor even large scale storage of learning data. Both effects complicate the statistical techniques required, and in particular make in-network decision-making and machine learning more difficult.

Mathematically, the network must process a data stream presenting non-stationary statistical characteristics. The various nodes in the WSN have different views on the phenomena being observed: some will be independent, others will be correlated, depending both on physical placement and environmental influences. The problem is then to subject this compound system to statistical analysis and develop a model that can autonomously manage and improve the quality of the information being deduced from the raw data streams, which can then either be returned for analysis in the cloud or used for local decision-making.

Sensor data usually exhibits strong spatial correlation: sensor readings measured at close distance are more similar. For example, at a specific time point s, we should not expect two temperature readings, $t_{i,s}$ and $t_{j,s}$, measured at co-located positions, i, j, to deviate significantly. In general, the spatial correlation assumption holds true for most sensor applications, as long as the underlying physical process over the space is continuous.

In this paper, we introduce a novel on-line in-network sensor fault detection algorithm that accounts for spatial correlation. The algorithm autonomously selects nodes to be used to verify observed readings, and uses dynamic learning to construct and maintain spatial models of the expected observational correlation. This solution has the following important features.

1. The proposed algorithm performs formal, well-founded fault detection but remain lightweight enough to be carried out locally on resource-constrained sensors. (We have implmented it on TMote Sky nodes.)

2. The solution features on-line fault detection performed locally in real time, which allows timely human intervention when sensor failure is detected. The fault detection algorithm is combined with normal data collection protocols so that the extra communication for fault detection is minimised.

3. The learning algorithm is robust to noisy sensor data, working even when faults are present in the training data.
4. The model adapts to the changes in the underlying physical process, so that false positives originating from stale models are minimised.

The structure of the paper is as follows. In section 2 we first present the spatial assumption model used for the algorithm and introduce the fault detection technique in detail in the next section. The combination of the technique with existing data collection protocol as well as implementation details are illustrated in section 4. The performance evaluation of the proposed algorithm is presented in section 5. We briefly compare our approach to other related work in section 6 before concluding in section 7 with some future directions.

2 Model Assumptions

Spatial Correlation Model

We use $\{Y_{i,t}\}$ to denote a time series of observations for $t \geq 0$ of some physical variable Y at location i. Given two series $\{Y_{i,t}\}$ and $\{Y_{j,t}\}$ of the same phenomenon observed from two locations i and j, we assume that both are unbiased observations of some true signal plus Gaussian white noise:

$$Y_{i,t} = \mu(x_i, t) + \epsilon_t, \quad \epsilon_t \overset{iid}{\sim} \mathcal{N}\left(0, \sigma_1^2\right) \tag{1a}$$

$$Y_{j,t} = \mu(x_j, t) + \varepsilon_t, \quad \varepsilon_t \overset{iid}{\sim} \mathcal{N}\left(0, \sigma_2^2\right), \tag{1b}$$

where $\mu(x, t)$ represents the ground truth of the underlying physical process at the location of x and time instance t.

For co-located sensors i, j, we can further assume $\mu(x_i, t) \approx \mu(x_j, t)$, as their distance d_{ij} is close. The approximate equality is valid as long as the physical process is diffusive and continuous. Without loss of generality, we assume the true signal difference is a small constant $\delta_{ij} = \mu(x_i, t) - \mu(x_j, t)$.

The spatial model simply states that for any two co-located sensor series $\{Y_{i,t}\}$ and $\{Y_{j,t}\}$, the synchronised difference $\{e_t = Y_{i,t} - Y_{j,t}, t \geq 0\}$ is a Gaussian random variable. That is to say,

$$e_t \overset{iid}{\sim} \mathcal{N}(\delta_{ij}, \sigma_e^2). \tag{2}$$

The model can be easily verified given (1).

Note that the variance σ_e^2 can be treated as a known constant in most WSN context, since $\sigma_e^2 = \mathrm{Var}[\epsilon - \varepsilon] = \sigma_1^2 + \sigma_2^2$, where σ_i is the measurement uncertainty associated with the sensor at node i and which can typically be read-off the component data sheets. For example, the SH10 temperature sensor installed on the TMote Sky nodes offers an accuracy of ±0.5℃, leading to a pairwise measurement uncertainty of $\sigma_e = \sqrt{2 \times 0.5^2} = 0.7$℃.

System Assumptions

The following assumptions are made in addition to the spatial correlation assumptions from above:

- We consider monitoring applications of WSNs, which require designated sensor nodes continuously to send back real-time sampled sensor data to one or more sinks. We assume that an appropriate routing protocol exists that can direct message flows from each local node to a sink.
- The network topology is assumed to be set in advance and known by local sensor nodes. This assumption is usually true for static WSN deployments, although not for, for example, air-deployed networks.
- Local synchronisation is maintained by local sensor clusters. Time synchronisation can be easily achieved by adding an integer type value into radio messages [1].

It is also advantageous (although not required) that the specification of the hardware – the mote and sensor chips – used in the deployment is known beforehand. However, the relevant parameters can be estimated from sensor data if no such description is available.

3 Statistical model

3.1 Two Inference Problems

The proposed solution revolves with two sub-problems. The first is verifier selection: each source node needs to find its qualified verifier nodes in the sense that valid spatial correlation is held between their data traces. Second, after the appropriate verifiers are selected, how the sampled sensor data at source node can be validated. We are going to show these two problems can be solved as probabilistic inferences by formal statistical tests.

Verifier Node Selection

The problem can be defined as follows: Given a sensor network that continuously monitors a continuous physical variable, for each node i with physical neighbours $Nbr(i)$, design a distributed algorithm to find a subset $Vrf(i) \subseteq Nbr(i)$ such that the spatial correlation is valid.

The reason for including this process is that, for some exceptional cases, geometrically co-located sensors may not necessarily exhibit spatial correlation. For instance, when one of co-located nodes is in a separate enclosure like another room, the readings reported may be quite different from its neighbours.

Based on the spatial assumption model, δ_{ij}, the true signal difference between two co-located sensors, should be a small quantity for spatial correlated nodes, i.e. they are essentially measuring the same phenomenon with marginal difference. Therefore, to filter out irrelevant nodes, one only needs to make inference on the size of δ_{ij}. Formally we set $\mathbb{P}(|\delta_{ij}| < \Delta_\mu | e_{1:n}) \geq 0.95$ for some predefined small constant Δ_μ in order to get the standard 95% confidence interval. In Bayesian terms, the distribution, $\delta_{ij}|e_{1:n}$, is called the *posterior distribution*.

Fault Detection

After the verifier node selection step, each source node i now has the knowledge of its data verifier set $Vrf(i)$. The problem of fault detection can be formed as follows: For a source node i and collected sensor reading $Y_{i,t}$, given its verifiers $Vrf(i)$ and their sensor readings $Y_{j,t}, j \in Vrf(i)$, design a distributed data validation algorithm to test whether $Y_{i,t}$ is sensor data fault or not.

To validate new sensor readings, instead of making inference on δ_{ij}, the hidden physical signal, we are going to make inference on the new observation $e_{n+1} = Y_{i,n+1} - Y_{j,n+1}$ directly. The distribution of e_{n+1} given historical data set $e_{1:n}$, i.e. $e_{n+1}|e_{1:n}$ is called *predictive distribution*. Based on it, the fault detection problem can be solved by calculating the probability of observing the new data: if the chance is small then it should be classified as a fault and vice versa.

In summary the solution boils down to two steps:

1. learn the relevant probability distributions; and then

2. solve the problems by forming statistical tests.

The detailed tests which depends on the distributions, are presented later in section 3.3 after the model learning algorithms are introduced.

3.2 Learning model

Efficient Online Learning

In this section, we are going to show the two distributions can be learnt in a online sequential way with minimum computational overhead and constant storage requirement, which is desirable for resource constrained devices like sensors.

According to Bayesian statistics, one can show that, by using uninformative prior [2, 3, 4]:

Case 1: σ_e is known

1. The posterior is
$$\delta_{ij}|e_{1:N} \sim \mathcal{N}\left(\hat{\mu}, \frac{\sigma_e^2}{N}\right); \tag{3}$$
2. The prediction is
$$e_{N+1}|e_{1:N} \sim \mathcal{N}\left(\hat{\mu}, (N+1)\frac{\sigma_e^2}{N}\right); \tag{4}$$

where $\hat{\mu} \triangleq \frac{1}{N}\sum_{i=1}^{N} e_i$.

Case 2: σ_e is unknown

1. The posterior distribution is Student T distributed with mean $\hat{\mu}$, variance $\frac{s^2}{N}$, and $N-1$ degree of freedom
$$\delta_{ij}|e_{1:N} \sim \mathcal{T}(\hat{\mu}, \frac{s^2}{N}, N-1); \tag{5}$$
2. The prediction is
$$e_{N+1}|e_{1:N} \sim \mathcal{T}(\hat{\mu}, (N+1)\frac{s^2}{N}, N-1); \tag{6}$$

where $s^2 \triangleq \frac{1}{N-1}\sum_{i=1}^{N}(e_i - \hat{\mu})^2$

To put this another way, we predict the expected signal difference and noise-driven error using a distribution learned in the previous N steps. Note that Case 2 distributions share the same mean but use an estimator s^2 instead of the known variance, comparing with Case 1. For both cases, the predictive distributions are identical to their posterior counterparts except the inflated variances (by a factor of $N+1$). Therefore, one only needs to learn the shared parameters once to obtain the two distributions.

It can be shown that the distributions for both cases can be learnt efficiently with linear growth time complexity and constant memory storage. Theorem 1 shows this claim.

Theorem 1 (Efficient Model Learning)**.** *The posterior distribution $\delta_{ij}|e_{1:N} \sim \mathcal{T}(\hat{\mu}, \frac{s^2}{N}, N-1)$ can be learnt in an on-line fashion with space complexity $\Theta(1)$, and time complexity $\Theta(N)$ via the following recursive procedure:*
$$\bar{\mu}_n = \bar{\mu}_{n-1} + \frac{1}{n}(e_n - \bar{\mu}_{n-1}), \tag{7a}$$
$$S_n = S_{n-1} + (e_n - \bar{\mu}_{n-1})(e_n - \bar{\mu}_n),, \tag{7b}$$
$$s^2(n) = \frac{S_n}{n-1} \tag{7c}$$
and
$$\hat{\mu} = \bar{\mu}_N \quad \textit{and} \quad \frac{s^2}{N} = \frac{S_N}{N(N-1)}. \tag{8}$$

Proof. By defining $\bar{\mu}_1 = e_1$ and $S_1 = 0$, for $1 < k \leq N$, $\bar{\mu}_k$ and S_k can be calculated at constant cost as a sum of $\bar{\mu}_{k-1}$, S_{k-1} and an adjusting term involving e_k. The time complexity is $\Theta(N)$ for $k = N$. Throughout the process, three parameters $\bar{\mu}_k, S_k$ and e_k are maintained, so the space complexity is of $\Theta(1)$. The proof of the given recursive procedures is omitted due to space limit. □

The learning formula for S_n is due to Knuth [5]. There are other one-pass algorithms for computing sample variance, notably

$$S_n = \sum_{i=1}^{n} e_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} e_i\right)^2$$

However, this equation needs to maintain two additional parameters in memory and then perform the subtraction of two substantial sums, which risks larger relative error.

Robust Learning by Markovs' Inequality

Just like normal sensor data, the learning data is subject to faults as well. We believe the assumption of error free learning data is not realistic; therefore, an additional step of error data filtering is added to make the algorithm robust. The filtering test applies Markov's inequality.

Theorem 2 (Markov's Inequality)**.** *Let ξ be a non-negative random variable and its mean $\mathbb{E}[\xi]$ exists, For any $t > 0$*

$$\mathbb{P}(\xi > t) \leq \frac{\mathbb{E}[\xi]}{t} \tag{9}$$

Proof. Since $\xi > 0$,

$$\begin{aligned}\mathbb{E}[\xi] &= \int_{-\infty}^{+\infty} \xi p(\xi) d\xi = \int_{0}^{+\infty} \xi p(\xi)\, d\xi = \int_{0}^{t} \xi p(\xi) d\xi \\ &+ \int_{t}^{+\infty} \xi p(\xi)\, d\xi \geq \int_{t}^{+\infty} \xi p(\xi) d\xi \geq t \int_{t}^{+\infty} p(\xi)\, d\xi \\ &= t\,\mathbb{P}(\xi > t).\end{aligned}$$

□

Let $\xi = (e_i - \bar{\mu}_n)^2$, then its mean $\mathbb{E}[\xi]$ can be estimated by $\bar{\mu}_\xi = \frac{S_n}{n} = \frac{1}{n}\sum_{i=1}^{n}(e_i - \bar{\mu}_n)^2$. When a new learning data entry e_k is arrived, the probability of observing some value as extreme as it is $P(\xi > \xi_k)$ whose value is smaller than ${}^{\bar{\mu}_\xi}/_{\xi_k}$, where $\xi_k = (e_k - \bar{\mu}_{k-1})^2$. By selecting a test probability threshold like $p_{thred} = 1\%$, we can filter out the noisy learning data. Note that the filtering test is lightweight and does not require any extra parameter estimation, as the only parameter used is S_n. The model learning algorithm is summarised in Figure 4.

3.3 Statistical Tests

Verifier Node Selection Test

As discussed in Section 2, to find spatial correlated nodes, one only needs to make inference on δ_μ by calculating the probability $\mathbb{P}(|\delta_\mu| < \Delta_\mu | e_{1:N})$ and comparing it with some predefined confidence level. The exact probability can be calculated by integration. However, to ease the computation, we instead derive the following test rule.

Theorem 3 (Verifier Node Test)**.** *If*
Case 1: $\delta_{ij}|e_{1:N} \sim \mathcal{N}\left(\hat{\mu}, \frac{\sigma_e^2}{N}\right)$,

$$\hat{\mu} + t_{\alpha,\infty}\sqrt{\frac{\sigma_e^2}{N}} < \Delta_\mu \quad \text{and} \quad \hat{\mu} - t_{\alpha,\infty}\sqrt{\frac{\sigma_e^2}{N}} > -\Delta_\mu, \tag{10}$$

Case 2: $\delta_{ij}|e_{1:N} \sim \mathcal{T}(\hat{\mu}, \frac{s^2}{N}, N-1)$,

$$\hat{\mu} + t_{\alpha,N-1}\sqrt{\frac{s^2}{N}} < \Delta_\mu \quad \text{and} \quad \hat{\mu} - t_{\alpha,N-1}\sqrt{\frac{s^2}{N}} > -\Delta_\mu, \tag{11}$$

then

$$\mathbb{P}(|\delta_{ij}| < \Delta_\mu | e_{1:N}) \geq 1 - 2\alpha, \tag{12}$$

where Δ_μ is a positive constant, and $t_{\alpha,N-1}$ is the critical percentile value from the corresponding Student T distribution. $t_{\alpha,\infty}$ is the Gaussian counterpart.

Proof. The transformed random variable

$$\left.\frac{\delta_{ij} - \hat{\mu}}{\sqrt{s^2/N}}\right| e_{1:N} \sim \mathcal{T}(0, 1, N-1).$$

So $\mathbb{P}(-t_{\alpha,N-1} < \frac{\delta_{ij}-\hat{\mu}}{\sqrt{s^2/N}} < t_{\alpha,N-1}|\mathcal{D}) = 1 - 2\alpha$, leading to

$$\mathbb{P}\left(\hat{\mu} - t\sqrt{\frac{s^2}{N}} < \delta_{ij} < \hat{\mu} + t\sqrt{\frac{s^2}{N}}\middle|\mathcal{D}\right) = 1 - 2\alpha.$$

If (11) holds, then

$$\begin{aligned}\mathbb{P}(|\delta_{ij}| < \Delta_\mu|\mathcal{D}) &= \mathbb{P}\left(\hat{\mu} - t\sqrt{\frac{s^2}{N}} < \delta_{ij} < \hat{\mu} + t\sqrt{\frac{s^2}{N}}\middle|\mathcal{D}\right)\\ &+ \mathbb{P}\left(-\Delta_\mu < \delta_{ij} \leq \hat{\mu} - t\sqrt{\frac{s^2}{N}}\middle|\mathcal{D}\right)\\ &+ \mathbb{P}\left(\hat{\mu} + t\sqrt{\frac{s^2}{N}} \leq \delta_{ij} < \Delta_\mu\middle|\mathcal{D}\right)\\ &\geq 1 - 2\alpha.\end{aligned}$$

By the converging property for Student T distribution, the proof for case 1 follows by setting $N = \infty$. □

According to Theorem 3, there are three user controlled parameters: the learning data size N, critical percentile value $t_{\alpha,N-1}$ and predefined spatial difference threshold Δ_μ. Some general rule of thumbs can be applied to select them. For example, confidence intervals of 95% and 90% ($\alpha = 0.025$ or 0.5) are commonly used in statistical tests. Learning data size should be relative large number, such as $N \geq 500$, which implies a more stable learnt model but also the Student T distribution converges to a Gaussian distribution: the corresponding Gaussian critical values can be used instead. The spatial difference threshold Δ_μ is used to specify how close co-located sensor readings are. Some expert field knowledge or even common sense can guide the selection of the value. For example, for a normal sensor application with average node distance of 10 meters, we should not expect two co-located temperature readings differ by 1℃.

Data Fault Test

The following theorem presents the statistical test to find out whether a new observation is faulty or not. A positive result from the test implies the probability of observing a value as extreme as the current one is small. The theorem actually rephrases a regular Student t-test, whose proof is omitted.

Theorem 4 (Data Fault Test)**.** *Given observations $Y_{i,N+1}$, $Y_{j,N+1}$ from sensor i, j, $\Delta_{N+1} = Y_{i,N+1} - Y_{j,N+1}$, and node $j \in Vrf(i)$. If*
Case 1: $e_{N+1}|e_{1:N} \sim \mathcal{N}\left(\hat{\mu}, (N+1)\frac{\sigma_e^2}{N}\right)$,

$$\hat{\mu} + t_{\alpha,\infty}\sqrt{(N+1)\frac{\sigma_e^2}{N}} < \Delta_{N+1} \quad \bigvee \quad \hat{\mu} - t_{\alpha,\infty}\sqrt{(N+1)\frac{\sigma_e^2}{N}} > \Delta_{N+1}, \quad (13)$$

Case 2: $e_{N+1}|e_{1:N} \sim \mathcal{T}(\hat{\mu}, (N+1)\frac{s^2}{N}, N-1)$,

$$\hat{\mu} + t_{\alpha,N-1}\sqrt{(N+1)\frac{s^2}{N}} < \Delta_{N+1} \quad \bigvee \quad \hat{\mu} - t_{\alpha,N-1}\sqrt{(N+1)\frac{s^2}{N}} > \Delta_{N+1}, \quad (14)$$

then

$$\mathbb{P}\left(|e_{N+1} - \hat{\mu}| > |\Delta_{N+1} - \hat{\mu}||e_{1:N}\right) \leq 2\alpha. \quad (15)$$

Multiple Verifiers Test For multiple verifiers, we can collect verification information from all (or a sub-set) of $vrf(i)$: if at least one verification result supports the suspect data, we mark the data as non-faulty. This mechanism protects against the risk of the breakdown of pairwise spatial correlations, since the synchronised difference e_t is only *partially stationary*, meaning that, which e_t remains stationary locally, it still evolves over the long term resulting in a learned historic model breaking down over time. It is substantially less likely that a node's observation will be *totally* different from those of *all* its neighbours. The following test rule is used.

$$faulty: \quad \text{if} \bigwedge_{j=1}^{vrf(i)} bool_j(Y_{i,t}) == true \tag{16}$$

3.4 Spatial model update

After the model $e_{N+1}|\{Y_{i,t}\}$ is learned initially from the first batch of N observations, it needs to be updated as more data are observed. There are essentially three reasons for updating:

1. The spatial difference e_t is only partially stationary, and so will evolve over time and invalidate the learned model;
2. Future data provide improved temporal correlations, allowing inference on the whole observed data series rather than only on the learned prefix; and
3. The computational effort involved in updating the model is significantly less than that involved in learning a new model from scratch if the initial model becomes outdated.

We further argue that only the mean estimator $\hat{\mu}$ needs to be updated, while the variance s^2 can be ignored. Under our spatial assumption, σ_e representing the measurement of independent sensors does not change with time: the performance of a "healthy" sensor neither improves nor degrades with time. Real world sensor data series can also demonstrate this claim. Figure 1 shows an excerpt of sensor traces from [6]: the grey line is the difference of two spatially correlated temperature sensor data, while the red line shows its rolling variance calculated with window size 100. Obviously, the variance remains constant comparing the sensor data. Secondly, although sensor performance may degrade over time in long run, the predictive distributions here are used as a reference model: expected readings from a normal sensor. Only comparing with a "healthy" normal model, faulty data can be found out. Finally, dropping an unnecessary regular parameter update can bring in the benefit of energy conservation.

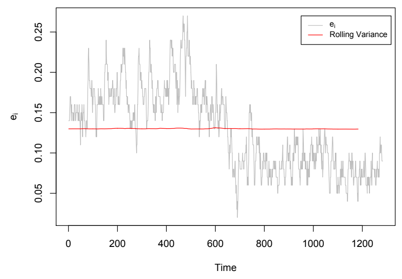

Figure 1: Spatial difference sensor data e_i with its mean adjusted rolling variance.

Obviously only data classified as non-faulty should be admitted into the model as it is updated. We reuse the test in Theorem 4 to check the eligibility of a data entry e_t to be incorporated into the model. However, a different, or more conservative, critical value t_α should be used. The motivation is to protect the model update from over data selection. For example, if 90% confidence interval was used, then there would be approximately 10% of good data being excluded from update. To differentiate these two thresholds, we denote the significance level used for update selection as α_{update} and α_{ftest} for the other.

Temporally close observations should resemble each other in the same way as spatially close observations. Another way of looking at this is that different elements of $\{Y_{i,t}\}$ carry different amounts of information. Therefore, for prediction, each data entry in $e_{1:N}$ carries a different level of information. It is natural to give them varying weights based on this correlation.

Note the model update procedure for the mean in Theorem 1, however, gives equal weights. To see this, the formula can be rewritten as

$$\begin{aligned}\bar{\mu}_n &= \frac{n-1}{n}(\frac{n-2}{n-1}\bar{\mu}_{n-2} + \frac{1}{n-1}e_{n-1}) + \frac{1}{n}e_n \\ &= \frac{1}{n}e_n + \frac{1}{n}e_{n-1} + \frac{1}{n}e_{n-2} + \dots\end{aligned}$$

We use the following recursive formula instead to update the model to take temporal correlation into account. The modified procedures provide estimators with time varying weighting such that newer data entries are given higher weights.

$$\tilde{\mu}_n = \tilde{\mu}_{n-1} + \psi(e_n - \tilde{\mu}_{n-1}) \tag{17}$$

where $0 < \psi < 1$ is called the smoothing parameter. The initial values can be simply set as the parameters learnt from the verifier selection step, i.e. $\tilde{\mu}_0 = \bar{\mu}_N$.

Note that (17) can be rewritten as $\tilde{\mu}_n = \psi e_n + (1-\psi)\tilde{\mu}_{n-1}$. By induction, it can be shown that

$$\begin{aligned}\tilde{\mu}_n &= \psi e_n + (1-\psi)\psi e_{n-1} + \ldots + (1-\psi)^n e_0 \\ &= \sum_{i=1}^{n} (1-\psi)^{n-i}\psi e_i + (1-\psi)^n e_0 = \sum_{i=0}^{n} w_i e_i,\end{aligned}$$

where $w_0 = (1-\psi)^n$, $w_i = (1-\psi)^{n-i}\psi$ for $1 \le i \le n$. The time varying weighting is evident as the weight decays exponentially as time traces back. Note that $\tilde{\mu}_n$ is a convex combination of the observations, i.e. the weights sum to one, which makes $\tilde{\mu}_n$ an unbiased estimator.

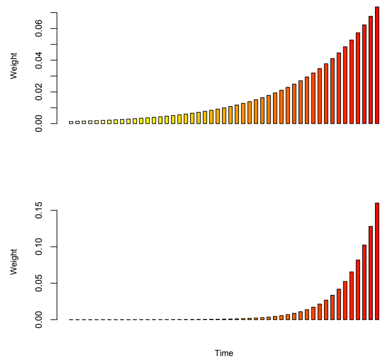

Figure 2: Time Varying Weights with $\psi = 0.1$ (top) and $\psi = 0.3$ (bottom) respectively.

The smoothing constant, ψ, is a user controlled parameter. As can be seen from Figure 2, large ψ, leading to a lighter tail, will give recent observations heavier weights. In other words, different ψ will make the system responsive to physical process changes at different rates. To help user specify the value, the following heuristic rule is derived.

The sum of partial weights on the k recent observations is

$$\begin{aligned}\sum_{i=n-k+1}^{n} w_i &= \psi \frac{1-(1-\psi)^k}{1-(1-\psi)} \\ &= 1-(1-\psi)^k;\end{aligned}$$

Therefore, the weights over the first $n-k$ historic data are given by

$$W(\psi) = (1-\psi)^k.$$

One only needs to choose a ψ such that $W(\psi)$ is small. For example, consider an application in which deployed sensors sample ambient temperature every

15 seconds. It is safe to assume observations within 10 minutes lag are more temporal correlated; so we should set $k = 40$. Therefore, selecting $\psi = 0.3$ will make sure $W(\psi) \approx 0$.

4 Protocol Design and Implementation

In this section, we present how the technique is incorporated into existing WSN data collection protocols.

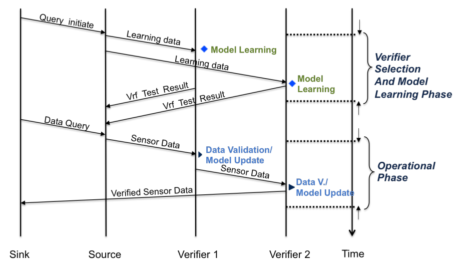

Figure 3: An Overview of the Message Passing Sequence.

A sequence diagram showing the whole life cycle of the proposed solution is listed in Figure 3. The whole data collection procedure starts with a learning phase, in which the objective is to learn the two distributions, which later will be used for the verifier selection test and fault detection in operational phase. More specifically, the relevant source nodes broadcast its sensor readings to its one-hop distance neighbours as learning data. Each potential verifier then learns the corresponding spatial model *via* (7). After a model with a predefined amount of learning data is learnt, each potential verifier carry out the verifier selection test by Theorem 3. The test result, if positive, is sent back to the corresponding source node. Upon this point, each source node and its corresponding eligible verifier nodes have established their source-verification relationships. Note that we differentiate source node and verifier node here only for the sake of clarity; however, each sensor can serve as both roles at the same time. Figure 4 summarises the steps involved in the learning phase.

The operational phase, in which the actual data collection takes place, is ensued. In group validation scenario, each sampled data entry, before sending back the sink, will be verified sequentially by its corresponding verifiers via the test discussed in Theorem 4. This is achieved via a simple routing protocol maintained by the source node. The source node inserts a FIFO queue of verifier addresses in the order of the desired sequence into the radio message (shown in Section 4). The verifier node whose address matches the destination verifies the contained data. Based on the test result, the current verifier can sets the

```
Input: $msg$: received learning data message
1: if Packet.src($msg$) $\in$ $Nbr$ then    ▷ If the source node is within one hop distance
2:     $e_i \leftarrow msg.sensorData - localData$
3:     $p \leftarrow {}^{n \times e_i^2}/_{S_n}$    ▷ Robust learning using Markov's inequality, see Theorem 2
4:     if $p \leq p_{thred}$ then
5:         discard $e_i$
6:     else
7:         update corresponding parameters via Theorem 1
8:     end if
9: end if
10: if $n == LearningDataSize$ then    ▷ Learning phase finishes
11:     verifier node test via Theorem 3
12:     if test is positive then
13:         notify source node    ▷ Notify the source node if test is positive
14:     else
15:         discard corresponding model    ▷ Delete the model parameters to save space
16:     end if
17: end if
```

Figure 4: Learning Spatial Model

```
typedef nx_struct VrfRadioMsg {
  nx_uint16_t id; /* Node id of source mote. */
  nx_uint16_t count; /* Epoch count */
  nx_uint8_t qIndex; /* Front of the vrf queue*/
  nx_uint8_t qSize; /* The size of the queue */
  nx_uint16_t vrfQueue[MAX_NBRSIZE]; /* Verifier addrs Queue*/
  nx_uint16_t voltage;
  nx_uint16_t sensorData; /* Sensor data from source*/
  nx_uint8_t vrfRst; /* A Boolean flags the vrf result so far */
} VrfRadioMsg_t;
```

Listing 1: Verification Message Layout

message destination either to the next verifier in the queue, or the sink, if the test result is negative, i.e. not faulty. The data message will finally be delivered to the sink by normal WSN routing protocols such as the Collection Tree Protocol [7]. The verifiers who test the data also need to update its local model according to (17). To give each verifier equal chance to update their local models and, more importantly, to balance the work load, it is advisable for the source node to change the verifier sequence queue from time to time by rotating the queue or even shuffle the sequence. The algorithm for data validation procedure is summarised in Figure 6.

Figure 5 shows an example of deployment featuring the proposed data validation technique. There are two source nodes, $S1$ and $S2$; each with its verifier node set $\{S2, V2\}$ and $\{V3, V4, V5\}$ respectively. Note that $S2$ serves as both source node and verifier node for $S1$ in this case. We use hollow arrows to represent the message passing for local data validation; while the regular arrows are used to mark the message relay to the sink. In this case, $S2$ sets its verification sequence as $[V3, V4, V5]$. However, the group validation mechanism finalises its decision before reaching $V5$; therefore, the verified data is enroute to the sink directly.

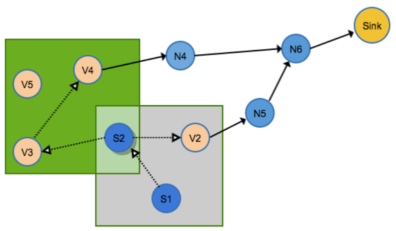

Figure 5: A WSN Deployment with Spatial Fault Detection

5 Evaluation

We access the solution in various aspects. First, to show the proposed algorithm is lightweight enough to run in sensors, we implement the solution and deploy it to sensor nodes. Second, numeric simulation is done to access the detection accuracy.

5.1 Implementation

To demonstrate the algorithm is a feasible solution for resource constrained sensors, we have implemented the framework in nesC on TinyOS 2.1.0 [8] and evaluated it using IEEE 802.15.4 complaint TMote Sky mote. It consists of an

```
Input: msg: a radio message of VrfRadioMsg_t received
Output: Check the contained data and forward the msg accordingly
 1: if Packet.isForMe(msg) then
 2:     (isFault, toUpdate) ←
            dataValidate(msg.sensorData)
                                                ▷ Data test via Theorem 4
 3:     if toUpdate == true then                ▷ Include for model update
 4:         updateModel(msg.sensorData)          ▷ Model update using (17)
 5:     end if
 6:     msg.qIndex ← msg.qIndex + 1
 7:     if isFault == true && msg.qIndex < msg.qSize then
                                  ▷ If it is a fault, forward to the next verifier
 8:         Packet.destination(msg) ←
                msg.vrfQueue[msg.qIndex]
 9:         Packet.send(msg)
10:     else                         ▷ Else flip the flag and send to the sink
11:         msg.vrfRst ← false
12:         Packet.destination(msg) ← root
13:         Packet.send(msg)
14:     end if
15: end if
```

Figure 6: Data Validation At a Verifier Node

Table 1: Program Size Comparison

	Without SDV	SDV (increase)	P. of total
RAM (in Bytes)	492	576 (17%)	5.6%
ROM (in Bytes)	15860	20872 (24%)	42.5%

processor running at maximum 8MHz and RAM of 10 KB [9]. The relative small RAM size becomes a major hindrance for the system and application program.

The footprint of a typical node which serves as both source and verifier is reported in Table 1. Comparing with a pure data collection implementation, a marginal increase of 17% in RAM with the spatial data validation (SDV) augmented solution is observed. Actually, the program size only depends on the number of spatial models kept locally, Figure 7 shows this relationship. Note that even for the extreme case of 64, i.e. a verifier serving 64 source nodes at the same time, the RAM is 696 bytes, only constituting 6.8% of the total memory (the ROM is 20876 bytes).

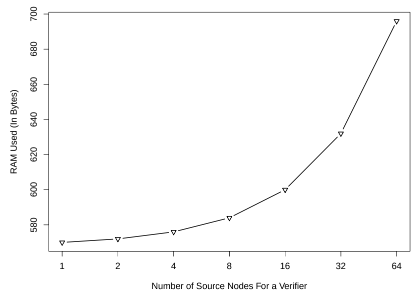

Figure 7: Memory Footprint of the Solution Versus the Number of Local Spatial Models

5.2 Numerical simulation

To better understand the detection accuracy of the solution, we use a real-world data set: the Intel Lab Data [6] to run numerical simulation. All the experimental results are obtained from simulations written in R [10]. The user controlled parameters used for the evaluation are listed in Table 2.

Fault Model Injecting artificial faults into a real data set is a common approach to measuring the accuracy of a detector [11, 12, 13]. For our evaluation, four particular kinds of faults are considered: short, constant, noise and drift.

Table 2: A Summary of Key Parameters

Name	Value Used	Description
N	Learning data size	500
p_{thred}	Probability threshold set to filter out erroneous learning data; see Section 3.2	1%
Δ_{μ}	Spatial difference threshold for co-located sensor readings	0.5℃
α_{vtest}	Significance level set for verifier selection test	0.25%
α_{ftest}	Significance level set for data validation test	0.25%
α_{update}	Significance level set for selected update test	0.05%
ψ	Decaying parameter set for time varying parameter update	0.3

Table 3 summarises the definitions, models and the parameters used for the different faults. The parameters are selected based on existing works [11, 12].

Detection Accuracy The faults detected mainly can be categorised into the following four classes: data points correctly detected as faulty (true positive, TP); data points correctly detected as non-faulty (true negatives, TN); data points incorrectly detected as faulty (false positives, FP); and data points incorrectly detected as non-faulty (false negatives, FN). Two simulation scenarios are considered:

1. Injecting errors only in source nodes;

2. Injecting errors in both a source node and its neighbours.

Two measurements, sensitivity and specificity, are reported. Note that

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{18}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{FP} + \text{TN}}. \tag{19}$$

Sensitivity, also called true positive rate, measures the correctly identified positive proportion; good sensitivity (approaching 1) means most of the faults are

Table 3: Different Fault Models

Class	Definition	Model	Parameters
SHORT	Momentary deviation from the true reading	$S_s(x,t) = g(x,t) + f * g(x,t)$	random f, i.e. fault intensity, from $[0.1, 10]$ is assigned
CONSTANT	Sensor readings remain constant for an unexpected period	$S_c(x,t) = c$, where $t \in T$	random c from $[33, 999]$ is chosen
NOISE	Sensor readings exhibit large unexpected variation	$S_n(x,t) = g(x,t) + N(0, \sigma^2)$, where $t \in T$	random σ from $[3, 10]$ is assigned
DRIFT	Sensor readings deviate from true values by a time-varying offset	$S_d(x,t) = g(x,t) + f(t)$, where $t \in T$ and $f(t) = a^t$	random a from $[2, e]$ is assigned

Table 4: Simulation Results of Detection Accuracy

		SHORT	CONST.	NOISE	DRIFT
CASE 1	Sens.	1.0	1.0	0.774	0.996
	Spec.	1.0	1.0	1.0	1.0
CASE 2	Sens.	1.0	1.0	0.682	0.996
	Spec.	0.999	0.999	0.999	0.999

Table 5: Comparing with PLA

	Sensitivity	Specificity
SDV	0.932	1.0
PLA (clean training data)	1.0	0.98
PLA	0.184	0.997

correctly identified. On the other hand, specificity measures the proportion of negatives, i.e. innocent sensor data, which are correctly identified as such; therefore, good specificity also means low false positive rate. According to Table 4, we observe the solution performs well for Short, Constant, and Drift fault types. It successfully detects almost all the faults but keeps the false positive rate low. For case two, the performance degrades a bit when faults are introduced to verifiers as well. However, because of the employed Multiple Verifiers Test, the risk of detection errors is shared among the verifiers, leading to a minor degradation. The relative poor performance of fault type Noise is due to the fact that the added noises are zero mean Gaussian samples, which lead to minor faults, i.e. deviating too marginally from the truth to be identified as a fault.

The good specificity means the solution has a very low false positive rates. As pointed in [13, 14], false positives are usually caused by mismatching detection models. As sensor data is always evolving, models learnt by historic data, if not updated, will not commensurate with the changing phenomenon, leading to false positives. The relative low FP rate is attributed to the employed time varying model update procedure, which makes the spatial model adaptive to the changing phenomenon. Figure 8 shows an example of this adaptiveness. The upper figure shows the evolution of a pair of sensor data; while the second figure shows the corresponding spatial difference e_t, and the solution successfully catches this evolution by updating its model parameter (the red line) *via* (17).

Learning in Noisy Environments The proposed solution uses a robust learning method to downplay the effects of faults in learning data. We evaluate

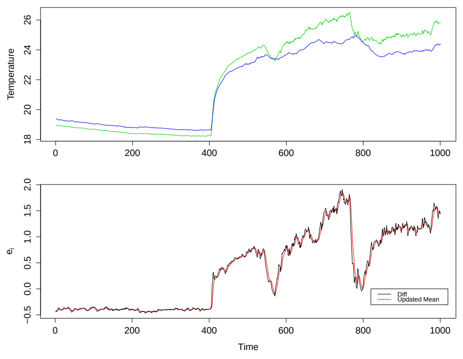

Figure 8: Time Varying Model Update

this effect by injecting errors into learning data set and compare the performance with the solution proposed in [11]. The method [11], called PLA here, employs a ad hoc heuristic rule to derive the faulty data threshold. As shown in Table 5, after introducing errors into learning data, the performance degrades in sensitivity (decrease 6.8%) while the false positive rate is still great. On the other hand, PLA suffers a more severe sensitivity degradation (over 81.6%) in comparison with its error free learning counterpart.

6 Related work

Ni [15] presents a detailed taxonomy of sensor data faults; however, no detailed detection method is proposed. Sharma [12] also studies data faults, as well as their possible causes. Four different data fault detection methods are compared: heuristic methods, estimation methods, time series analysis methods, and learning-based methods. The authors found that these four classes sit at different points on the detection accuracy spectrum. But none of the four are on-line and distributed solutions neither not adaptive. They depend heavily on domain/expert knowledge to set learning parameters beforehand rather than adjusting the learned model adaptively. A good survey on outlier detection techniques is presented by Zhang *et alia* [16], although most of the solutions work off-line.

A packet-level attestation method to increase sensor data reliability is proposed by Kamal [11]. This work attaches an attestation bit to each observation to indicate its validity by exploiting one-hop spatial correlation. However, it uses an ad hoc heuristic model without proper mathematical justification. Moreover, it ignores the possible breakdowns in the correlation between neighbouring sensor readings, which may result in wrong validations. One typical Naive Bayesian Network based solutions find outliers from a probabilistic viewpoint, but suffer from memory explosion [17]. An alternative solution [18] presents a on-line solution based on statistical tests and the technique can distinguish between erroneous measurements and events. However, no flexible adaptive procedure is given to update the model and no details on the integration of the solution and existing data collection protocol is presented.

7 Conclusion

We have proposed an online, distributed fault detection technique. The solution is lightweight and integrates well with existing data collection protocols for WSNs. To make the solution adaptive to the changing physical phenomenon, we make use of a time varying model update procedure. Simulation results show that the solution can effectively detect short, constant, and drift faults. We have also implemented the solution on real-world sensors, with a memory footprint less than 6% of the total available memory.

Some parts of the solution still need further investigation. For instance, due to time limits, we only implemented the core part of the solution and no comprehensive evaluation on the real-world deployment has been carried out. Because ground truth for regular sensor data is missing, most fault detection works are evaluated by injecting artificially-created faults. However, we think it would be better to directly apply the solution blindly to original sensor data and compare the result over different solutions. For example, we may compare the test result of our computationally cheap solution with more complex solutions featuring more sophisticated statistical models, like Gaussian process or Bayesian space-time models, computed at server-side. The sophisticated solutions can serve as reference models in the absence of ground truth.

References

[1] P. Levis and D. Gay, *TinyOS programming.* Cambridge University Press, 2009.

[2] D. S. Sivia, *Data analysis: a Bayesian tutorial.* Oxford university press, 1996.

[3] K. P. Murphy, *Machine learning: a probabilistic perspective.* MIT Press, 2012.

[4] G. Petris, S. Petrone, and P. Campagnoli, *Dynamic linear models with R.* Springer, 2009.

[5] D. E. Knuth, "The art of computer programming, 3rd edn., vol. 2," *Seminumerical Algorithms*, 1998.

[6] INTEL, "Intel Berkeley Laboratory sensor data set," http://db.csail.mit.edu/labdata/labdata.html, 2004.

[7] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems.* ACM, 2009, pp. 1–14. [Online]. Available: http://dl.acm.org/citation.cfm?id=1644040

[8] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "TinyOS: An Operating System for Sensor Networks," in *Ambient Intelligence*, W. Weber, J. Rabaey, and E. Aarts, Eds. Berlin/Heidelberg: Springer Berlin Heidelberg, 2005, ch. 7, pp. 115–148. [Online]. Available: http://dx.doi.org/10.1007/3-540-27139-2_7

[9] Moteiv, *Tmote Sky Datasheet http://www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf*, 2006.

[10] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2012, ISBN 3-900051-07-0. [Online]. Available: http://www.R-project.org

[11] A. R. M. Kamal, C. Bleakley, and S. Dobson, "Packet-Level Attestation (pla): A framework for in-network sensor data reliability," *ACM Trans. Sen. Netw.*, vol. 9, no. 2, pp. 19:1–19:28, Apr. 2013. [Online]. Available: http://doi.acm.org/http://dx.doi.org/10.1145/2422966.2422976

[12] A. Sharma, L. Golubchik, and R. Govindan, "Sensor faults: Detection methods and prevalence in real-world datasets," *ACM Transactions on Sensor Networks*, vol. 6, no. 3, pp. 23–33, 2010.

[13] J. Gupchup, A. Sharma, A. Terzis, A. Burns, and A. Szalay, "The perils of detecting measurement faults in environmental monitoring networks," in *Proceedings of the International Workshop on Wireless Sensor Network Deployments (WiDeploy)*, 2008.

[14] L. Fang, S. Dobson, and D. Hudges, "An error-free data collection method exploiting hierarchical physical models of wireless sensor networks," in *Proceedings of the 10th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, &#38; Ubiquitous Networks*, ser. PE-WASUN '13. New York, NY, USA: ACM, 2013, pp. 81–88. [Online]. Available: http://doi.acm.org/10.1145/2507248.2507255

[15] 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*, vol. 5, no. 3, pp. 25:1–25:29, 2009. [Online]. Available: http://doi.acm.org/10.1145/1525856.1525863

[16] Y. Zhang, N. Meratnia, and P. Havinga, "Outlier detection techniques for wireless sensor networks: A survey," *Communications Surveys Tutorials, IEEE*, vol. 12, no. 2, pp. 159–170, 2010.

[17] E. Elnahrawy and B. Nath, "Context-aware sensors," in *Wireless Sensor Networks*. Springer, 2004, pp. 77–93.

[18] L. Bettencourt, A. Hagberg, and L. Larkey, "Separating the wheat from the chaff: Practical anomaly detection schemes in ecological applications of distributed sensor networks," in *Distributed Computing in Sensor Systems*, ser. Lecture Notes in Computer Science, J. Aspnes, C. Scheideler, A. Arora, and S. Madden, Eds. Springer Berlin Heidelberg, 2007, vol. 4549, pp. 223–239. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-73090-3_15