Agnostic Diagnosis: Discovering Silent Failures in Wireless Sensor Networks

Xin Miao, Kebin Liu, Yuan He, Dimitris Papadias, Qiang Ma, and Yunhao Liu

***Abstract*—In wireless sensor networks (WSNs), diagnosis is a crucial and challenging task due to the distributed nature and stringent resources. Most previous approaches are supervised, relying on a-priori knowledge of network faults. Our experience with GreenOrbs, a long-term large-scale WSN system, reveals the need of diagnosis in an agnostic manner. Specifically, in addition to predefined faults (i.e., with known types and symptoms), silent failures that are unknown beforehand, account for a large fraction of network performance degradation. Currently, there is no effective solution for silent failures because they are often diverse and highly system-related. In this paper, we propose Agnostic Diagnosis (AD), an online lightweight failure detection approach. AD is motivated by the fact that the system metrics (e.g., radio-on time, number of packets transmitted) of sensor nodes usually exhibit certain correlation patterns. Violations of such patterns indicate potential silent failures. We implement AD on a working WSN consisting of 330 nodes. Our experimental results demonstrate the advantages of AD to discover silent failures, effectively expanding the capacity and scope of WSN diagnosis.**

***Index Terms*—Diagnosis, sensor networks.**

I. Introduction

RECENT advances in Wireless Sensor Network (WSN) technologies have enabled various applications such as environmental surveillance, traffic monitoring and emergency navigation [1]–[4]. WSNs are by nature error-prone and have unsatisfactory reliability, encountering various faults and failures during their operation. Consequently, diagnosis has drawn substantial attention in recent years as a method to enhance the applicability, reliability, and efficiency of WSNs.

However, diagnosing WSNs is a challenging issue because, once a WSN is deployed, its inner conditions are not directly observable. Specifically, since many WSNs reside in harsh or remote environments, it is difficult to perform in-situ troubleshooting on the faulty nodes. Furthermore, the distributed nature and stringent resources of WSNs render it hard for a network operator to completely monitor the system's working status. Due to similar reasons, it is also infeasible to deploy management tools like SNMP, or other costly diagnostic modules, on the sensor nodes.

Many existing diagnostic approaches are supervised, i.e., they rely on either specific rules or inference models. An obvious drawback is that they are limited to faults with known types and symptoms, and hence, they cannot be easily generalized to different application scenarios. On the other hand, the interactions within the WSN and the causal dependencies between root causes and symptoms are usually unknown. As a result, silent failures remain undetected.

This paper is motivated by the need for long-term reliable operation of GreenOrbs, a large-scale WSN system in a forest [2]. Currently, GreenOrbs includes 330 nodes and has been in continuous operation for over eight months. During the deployment, we often observe system performance degradations, e.g., low packet delivery ratios. A portion of faulty nodes can be easily identified since they generate apparently abnormal system metrics (e.g., measurements that are clearly beyond the reasonable scope). The other faulty nodes, however, cannot be identified in this way. For instance, we adopt low-power listening mode so the radio is switched on only for receiving, sending, or idle listening. Consequently, the radio-on time should be closely correlated with the amount of traffic passing the node. We noted that during a five-minute period, a node kept its radio on for 47.5 seconds, transmitting only 3 packets in total. In the next five-minute period, it kept its radio on for 51.6 seconds and transmitted 550 packets. Any individual value of the metrics is not abnormal, but the correlation between radio-on time and number of transmitted packets clearly suggests inconsistency on that node.

A straightforward solution would be to develop a set of static correlation rules for identifying the faulty nodes. However, this is inapplicable for two reasons: (i) usually there is insufficient domain knowledge to enumerate all the rules; (ii) WSN deployment is often evolutional, so that the correlation rules change over time; both software upgrades and environment conditions may have a great impact on the correlations.

To overcome these problems, we propose Agnostic Diagnosis (AD), an online lightweight approach for WSNs. AD exploits the correlations among metrics of each sensor using a *correlation graph* that describes the latent status of the node. Such a correlation graph is updated periodically using the node's metrics. By mining the correlation graphs, we identify the underlying rules of a normally running system, and detect abnormal correlations.

Our main contributions are summarized as follows:

- Unlike previous approaches, Agnostic Diagnosis does not rely on predefined rules. It relies on minimal a-priori knowledge and thus can be applied to a wide variety of WSN applications.
- We propose the correlation graph, a compact structure

Manuscript received November 15, 2012; revised March 11 and June 30, 2013; accepted September 2, 2013. The associate editor coordinating the review of this paper and approving it for publication was N. Kato.

X. Miao, K. Liu, Y. He, Q. Ma and Y. Liu are with the School of Software and TNList, Tsinghua University, Beijing, China (e-mail: {miao, kebin, he, maq, yunhao}@greenorbs.org). K. Liu is the corresponding author.

D. Papadias is with the Department of Computer Science and Engineering, Hong Kong University and Technology, Clear Water Bay, Kowloon, Hong Kong (e-mail: dimitris@cse.ust.hk).

Digital Object Identifier

that efficiently characterizes the internal correlations inside a node.
- We implement AD and evaluate it with traces from a 330-node GreenOrbs deployment. Case studies and statistics demonstrate the effectiveness of AD.

The rest of the paper is organized as follows. Section 2 motivates the problem. Section 3 presents AD. Section 4 evaluates our design and Section 5 surveys related work. Finally, Section 6 concludes this paper.

II. Motivation

Diagnosis is a fundamental task for long-term large-scale WSN systems. This section first introduces the basic information and application requirements of GreenOrbs, including several observations in the form of concrete examples. These observations reveal the existence of correlation patterns among the nodes' operational metrics and the feasibility of AD.

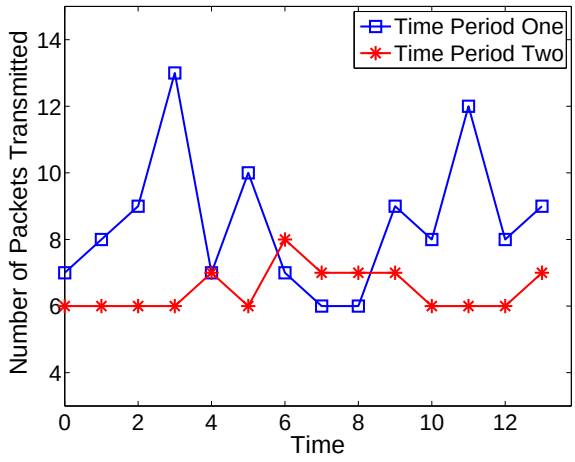
A. GreenOrbs

GreenOrbs is an ecological surveillance project deployed in a forest. It collects a group of sensory data such as temperature, humidity, illumination and carbon dioxide concentration to support various applications. Since the deployment is in a remote area, the overhead of in-situ debugging and troubleshooting is very high. Therefore, the diagnostic system is designed so that each node periodically transmits its current status to the sink. Specifically, we collect 22 types of metrics (Table I) from each node that are classified into four categories: (1) timing metrics e.g. RadioOnTimeCounter, which denotes the accumulative radio-on time; (2) traffic metrics, e.g. TransmitCounter, which records the accumulative number of packets transmitted by a node; (3) task metrics, e.g. TaskExecCounter, which is the accumulative number of tasks executed; (4) other metrics such as ParentChangeCounter, which counts the number of parent changes.

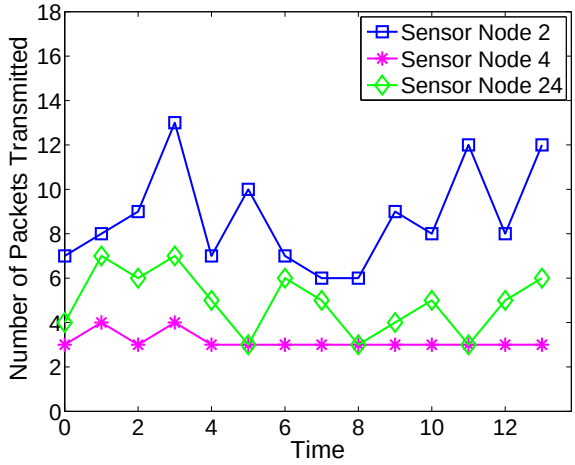
The difficulty of diagnosis in GreenOrbs stems from the absence of a-priori knowledge on the possible faults and their symptoms, which renders general diagnostic rules inapplicable. Instead, we design a diagnostic system requiring minimal domain knowledge.

B. Silent Failures

In this section, we demonstrate an example of silent failures that could hardly be discovered using rule-based methods, as well as an example that could be mistakenly flagged. Fig. 1a plots the number of packets transmitted by a node in two time periods. Its value is counted in every fifteen-minute time frame. For example, point (0,6) means the node transmitted six packets within the first fifteen minutes. During time period one, the metric changes dramatically, with its values ranging from 6 to 13. One would think that the node might be faulty in this time period. However, a manual analysis of the whole trace reveals that the node works well. On the other hand, during time period two, this metric appears to be stable, and the difference between maximum value and minimum value is only 2. Nevertheless, after manually checking the network trace, we find that the node actually encounters routing loops



(a) Number of packets transmitted by node 2 in two time periods



(b) Number of packets transmitted by sensor nodes 2, 4 and 24 in a same period

Fig. 1. Examples of normal operation and node failures.

(a fatal routing problem). This example demonstrates that the individual examination of metrics on the same sensor node may overlook silent failures, or may flag failures by mistake.

Fig. 1b shows another example. In this figure, the same metric is studied, but on different sensor nodes during the same time period. It shows that the curve of node 4 is flat, while curves of node 2 and 24 are steep. The reason why they have different curves is as the following: (1) First, node 2 and node 4 play different roles in the network. Node 4 is a leaf node, and it seldom has to forward packets for other nodes. So the number of packets transmitted is 3 most of the time. However, node 2 is an intermediate node. It has to forward packets for its children, so the number of packets transmitted is always above 6. (2) Second, radio irregularity may cause changes in the number of node 2's children. When node 2 has more child nodes, it transmits more packets. (3) Third, since nodes in the network are not synchronized, they wake up and sleep according to their local clock. As a result, traffic patterns in network do not remain stable. This may also cause fluctuations in the packets transmitted by node 2. Although the metric on node 4 exhibits a quite different pattern from those of the other

TABLE I
EXAMPLES OF SYSTEM METRICS

Metric Names	Meaning	Metric Names	Meaning
RadioOnCounter	number of times radio is turned on	DuplicateCounter	number of duplicated packets
RadioOnTimeCounter	radio-on time in milliseconds	SuccAckCounter	number of successfully transmitted packets
ReceiveCounter	number of packets received	ParentChangeCounter	number of parent changes
TransmitCounter	number of packets transmitted	NoParentCounter	number of times when no parent is found
SelfTransmitCounter	number of packets transmitted by the node itself	TaskPostCounter	number of tasks posted in TinyOS programs
RetransmitCounter	number of retransmissions	TaskExeCounter	number of tasks executed in TinyOS programs
LoopCounter	number of routing loops detected	TaskSendFailCounter	number of tasks failed in TinyOS programs

two nodes, in fact all three nodes perform well. This example suggests that even if considering multiple sensor nodes, an individual metric is still insufficient to uncover failures.

C. Observations

We conduct the correlation patterns of the 22 metrics and visualize them using gray-scale images (the details will be presented later). The cell (i, j) in the image denotes the correlation score between metrics i and j. Fig. 2(a)-(c) displays three images constructed from metrics of a same node. The node is normal in the first two periods, but faulty in the third one. Interestingly, the first two images are very similar to each other, whereas they differ significantly from the third one. Analogous results are observed in the spatial domain as well. Fig. 2(d)-(f) shows the images of three nodes in the same period. The first two nodes are normal, while the third one is faulty. Observe that the first two images follow a similar pattern, which is clearly distinguishable from the last one.

The difference of correlation patterns can be explained with the following example. There are two metrics called LoopCounter and SuccAckCounter. The former is the number of times that a same packet passes through the current node, and the latter is the number of packets that are successfully transmitted by the current node. If the node is normal, the value of LoopCounter should be zero most of the time. Thus, these two metrics have weak correlation. If the node is out of order and produces routing loops, the same packet passes multiple times. Eventually, LoopCounter dominates SuccAckCounter, and these two metrics are highly correlated, differentiating the faulty node from of the normal ones.

In summary, proper diagnosis of WSNs, especially long-term large-scale systems like GreenOrbs, is crucial and challenging. This is partially due to resource constraints and hardness to track in-network status. Even a more important, but often overlooked fact is that the diagnostic capacity is restricted by our incomplete knowledge of possible failures. Our experience with GreenOrbs reveals that independent investigation of metrics is insufficient for capturing all the problematic nodes. On the other hand, exploiting correlations of metrics uncovers significantly more failures.

III. AGNOSTIC DIAGNOSIS

We assume a WSN consisting of N sensor nodes. In each time frame t, a sensor s_i measures its working status, obtaining a status vector $S_{i,t} = (m_{1,t}, m_{2,t}, \ldots, m_{p,t})$, where p is the number of metrics and $m_{u,t}$ is the value of the u-th metric within time frame t, $1 \leq u \leq p$. Table II summarizes the symbols used in this paper as well as their meanings.

TABLE II
DEFINITION OF SYMBOLS

Symbols	Definition
N	number of sensors
p	number of metrics
w	window size; a time window consists of w time frames
s_i	sensor node i, $1 \leq i \leq N$
$S_{i,t}$	the p-dimensional status vector of sensor node i in time frame t, $S_{i,t} = (m_{1,t}, m_{2,t}, \ldots, m_{p,t})$
$m_{u,t}$	the value of the u-th metric in time frame t, $1 \leq u \leq p$
$c_k(u, v)$	correlation score of metrics u and v in time window k, $1 \leq u, v \leq p$
$CG_{i,k}$	correlation graph of sensor node i in time window k

A. Correlation Graphs

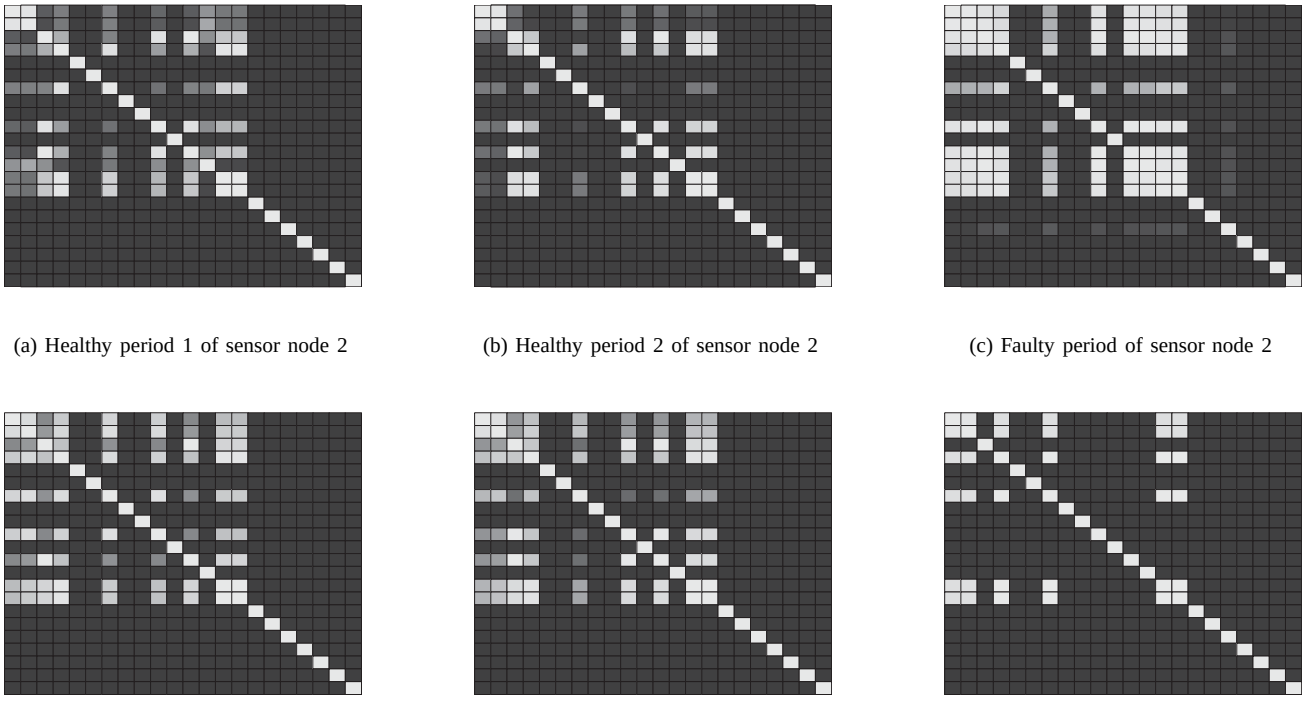
The correlation graph is a symmetric matrix representing the pair-wise correlations of system metrics:

$$\begin{pmatrix} c_k(1,1) & c_k(1,2) & \ldots & c_k(1,p) \\ c_k(2,1) & c_k(2,2) & \ldots & c_k(2,p) \\ \vdots & \vdots & \ddots & \vdots \\ c_k(p,1) & c_k(p,2) & \ldots & c_k(p,p) \end{pmatrix}.$$

It is constructed and maintained periodically, for each node in the WSN. The correlation between metrics are evaluated in each time window k, which lasts from time frame $(k-1)*w+1$ to time frame $k * w$, i.e., $[(k-1) * w + 1, k * w]$. Suppose $M_{u,k} = (m_{u,(k-1)*w+1}, m_{u,(k-1)*w+2}, \ldots, m_{u,k*w})$ and $M_{v,k} = (m_{v,(k-1)*w+1}, m_{v,(k-1)*w+2}, \ldots, m_{v,t*w})$ are the values of metrics u and v collected in time window k respectively. We define the correlation score between them using Pearson's product-moment coefficient:

$$c_k(u,v) = \frac{w \sum_{i=1}^{w} m_{u,(k-1)*w+i} m_{v,(k-1)*w+i}}{\sigma_{u,k}\sigma_{v,k}} - \frac{\sum_{i=1}^{w} m_{u,(k-1)*w+i} \sum_{i=1}^{w} m_{v,(k-1)*w+i}}{\sigma_{u,k}\sigma_{v,k}}, \quad (1)$$

where $\sigma_{u,k}$, $\sigma_{v,k}$ are the standard covariances of $M_{u,k}$ and $M_{v,k}$ respectively, This correlation score falls within the range $[-1, 1]$. The closer it is to either -1 or 1, the stronger the correlation between the variables is. As it approaches zero, the



(a) Healthy period 1 of sensor node 2 (b) Healthy period 2 of sensor node 2 (c) Faulty period of sensor node 2

(d) Healthy node 1 (e) Healthy node 2 (f) Faulty node

Fig. 2. Visualizations of correlation graphs.

correlation decreases. In Fig. 2, the visualization is achieved by transforming correlation scores into gray-scale values in the range of $[0, 255]$. Since the gray value of white is 255, lighter pixels correspond to stronger correlations.

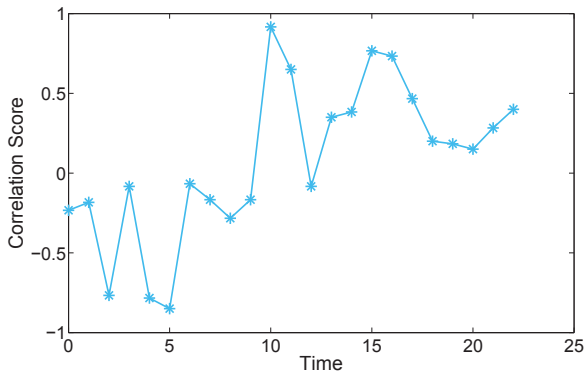
B. AD *Framework*

AD is a sink-based framework. It instruments the source code of TinyOS programs and thus is able to track the performance of programs using a number of performance counters, a.k.a, metrics. The values of these counters are important indicators of sensor nodes' performance. Thus they are required to be sent back to the sink periodically. On receiving the metrics from the network, AD starts to build correlation graphs in the current time window. After that, AD identifies anomalies in both temporal and spatial domain. Specifically, *temporal detection* refers to detecting sudden changes in the correlation graphs from a same node, while *spatial detection* discovers pattern inconsistencies using multiple nodes. If an anomaly is identified by both temporal and spatial detection, it has a high chance of signifying a real problem.

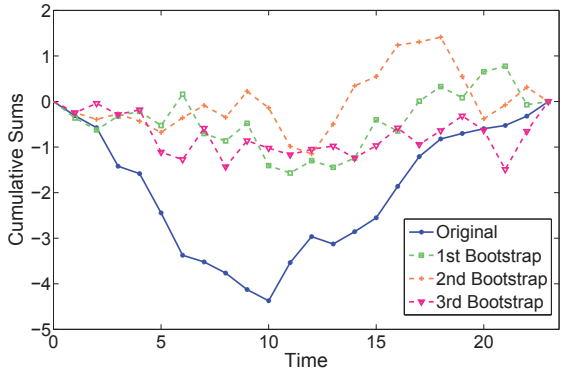
C. Temporal Detection

The problem of Temporal Detection is defined as the following.

Problem 1. (Temporal Detection) Given the correlation graphs $CG(i) = \{CG_{i,1}, CG_{i,2}, \ldots, CG_{i,t}, \ldots\}$ of node s_i, detect the time when an abrupt change happens in time series $\{c_1(u,v), c_2(u,v), \ldots, c_t(u,v), \ldots\}$, where $1 \leq u, v \leq p$ and $u \neq v$.



(a) Original time series



(b) Original cumulative sums and cumulative sums after bootstrap

Fig. 3. CUSUM examples.

Algorithm 1: Temporal detection algorithm

Input: (1) Correlation graphs of sensor node s_i, i.e., $\{CG_{i,t_{begin}}, CG_{i,t_{begin}+1}, \ldots, CG_{i,t_{end}}\}$;
(2) The confidence threshold $\epsilon_{threshold}$.
Output: The time t_{faulty} when s_i becomes faulty with confidence ϵ_{faulty}. $t_{faulty} = -1$ if s_i is not faulty.

```
for t = t_begin → t_end do
    Take the upper triangular part of CG_{i,t};
    Convert it into a column vector with length p(p−1)/2;
end for
for d = 1 → p(p−1)/2 do
    For each vector, take its d-th element;
    Obtain a new series {c_{t_begin}, c_{t_begin+1}, ..., c_{t_end}};
    (c_{t_cand}, ε_cand)=CUSUM(c_{t_begin}, c_{t_begin+1}, ..., c_{t_end});
    if ε_cand > ε_threshold then
        t_faulty ← t_cand;
        ε_faulty ← ε_cand;
    else
        t_faulty ← −1;
    end if
end for
```

Considering that the system maintains $p = 22$ metrics, for each node, there are totally $p \cdot (p-1)/2 = 231$ different time series. At the end of each time window, AD detects whether there is an abrupt change in each time series. The detection process is modeled as a change point analysis problem. We adopt a classical *CUSUM* algorithm [5] to discover change points. In our application, it includes three steps.

Step 1: Cumulative Sums Calculation. In this step, *CUSUM* charts are constructed based on the original time series. Let $\{c_1(u,v), c_2(u,v), \ldots, c_n(u,v)\}$ be the correlation scores between metrics u and v from the first time window to the current one. The cumulative sums $\{CS_0, CS_1, CS_2, \ldots, CS_n\}$ are calculated as:

- $CS_0 = 0$;
- $CS_i = CS_{i-1} + c_i(u,v) - \sum_{i=1}^{n} c_i(u,v)/n$.

The intuition behind cumulative sums is that if there are no abrupt changes in correlation scores, then the cumulative sums just shrink near zero. Otherwise, supposing at the beginning all the scores are above the average, the term $c_i(u,v) - \sum_{i=1}^{n} c_i(u,v)/n$ is always larger than zero, causing cumulative sums CS_i to increase steadily. If $c_{k+1}(u,v)$ is an abrupt change, CS_{k+1} should be much smaller than CS_k. Thus, CS_k will dominate both the preceding and the subsequent cumulative sums. The change score is denoted as $CS_{diff} = max(CS_i) - min(CS_i)$.

Step 2: Bootstrap Analysis. Bootstrap analysis provides a way to test the significance of the change by mimicking the behavior of *CUSUM* if there are no change points. In this step, the time series $\{c_1(u,v), c_2(u,v), \ldots, c_n(u,v)\}$ is reordered randomly. Based on the random ordered time series, a new sequence of cumulative sums $\{CS'_0, CS'_1, CS'_2, \ldots, CS'_n\}$ are computed. The new change score is $CS'_{diff} = max(CS'_i) - min(CS'_i)$. After performing bootsraps M times, among which there are X times that CS_{diff} is larger than CS'_{diff}, the confidence level of CS_{diff} is calculated using X/M. With sufficiently high M, we can accurately estimate the confidence. However, since each window has 10 data points, it is infeasible to iterate all the 10! cases. We found that $M = 1000$ is a reasonable value, for it is both large enough to accurately estimate the confidence and small enough to run the algorithm in a short period of time.

Step 3: Change Point Detection. Once the confidence level of CS_{diff} is higher than a predefined threshold, we say that an abrupt change happens in the current time series. We select the index k so that $CS_k = max|CS_i|$. Therefore, k is the last index before the change point $c_{k+1}(u,v)$. Through our experiments, we choose 90% as the confidence threshold to determine an abrupt change.

In Fig. 3a, the original time series has an abrupt change at index 10. In Fig. 3b, at the same position, the original CUSUM curve encounters a sudden change, which is consistent with the original time series. However, in the other CUSUM curves with random ordering, there is no sudden change, and all the curves tend to shrink near zero.

Algorithm 1 illustrates the details of temporal detection. This algorithm runs at the end of each time window, and uncovers the sensor nodes whose correlation graphs change acutely. It is possible that a node may experience multiple temporary failures, such that there are multiple change points in the CUSUM curve. We focus on the most significant change point because if it passes the confidence threshold, we can already say that the node is suspicious. Meanwhile, if it does not pass the confidence threshold, it means other change points will not pass the threshold either.

D. Spatial Detection

Spatial detection takes snapshots of correlation graphs of all nodes in each time window, and groups similar ones together. Sensor nodes whose correlation graphs diverge from the common patterns are considered suspicious.

Problem 2. (Spatial Detection) Given a set of nodes $\{s_1, s_2, \ldots, s_n\}$, as well as their correlation graphs in time window t, i.e., $\{CG_{1,t}, CG_{2,t}, \ldots, CG_{n,t}\}$, divide them into K clusters with cluster centers $C_1, C_2, \ldots, C_K$. The confidence level for a sensor node s_i to be suspicious is defined as: $\min_j(dist(CG_{i,t}, C_j))$, where $dist(CG_{i,t}, C_j)$ is a binary distance function between two correlation graphs.

Considering the symmetry of the correlation graph, we can take the upper triangular part of it and transform it to a d-dimensional vector, where $d = p \cdot (p-1)/2$. Therefore, an intuitive solution is to apply K-Means clustering [6] directly and find the farthest nodes from the centers. However, this approach is ineffective due to the high dimensionality of correlation graphs. In high dimensions, the concept of distance becomes meaningless [7], causing the clustering results to be imprecise. Another problem is the dependencies among dimensions. For instance, suppose metrics u, v, and w are highly correlated pair-wisely. As a result, the three correlation scores $c(u,v)$, $c(u,w)$, $c(v,w)$ are very likely to be dependent, suggesting that the actual dimensionality of the correlation graphs is much smaller than d.

Algorithm 2: Spatial detection algorithm

Input: (1) Correlation graphs of all sensor nodes in time window t, i.e., $\{CG_{1,t}, CG_{2,t}, \ldots, CG_{N,t}\}$;
(2) Dimensionality m ($m \ll p(p-1)/2$).

Output: A ranked list of sensor nodes sorted by the probability of being faulty.

1: **for** $i = 1 \rightarrow N$ **do**
2: Take the upper triangular part of $CG_{i,t}$;
3: Convert it into a row vector X_i with length $p(p-1)/2$;
4: **end for**
5: Construct the data matrix $X = [X_1, X_2, \ldots, X_n]$;
6: Compute the m largest eigenvalues of matrix $X^T X$;
7: Obtain the corresponding eigenvectors v_1, v_2, ..., v_m;
8: Construct a new data matrix $Z = (v_1\ v_2\ \ldots\ v_m)^T$;
9: Z_i, the i-th column of Z, is the projection of correlation graph $CG_{i,t}$ in m-dimensional space;
10: Cluster the set $\{Z_i\}$ using K-Means algorithm;
11: The cluster centers are C_1, C_2, ..., C_K;
12: **for** $i = 1 \rightarrow N$ **do**
13: The probability that sensor node s_i is faulty $\leftarrow \min_j(dist(b_i, C_j))$;
14: **end for**
15: Return the list of sensor nodes sorted by their probabilities of being faulty.

Following the idea of PCA, our algorithm first projects correlation graphs to a low-dimensional space and then clusters the projected data [8]–[10]. Let $X_1, X_2, \ldots, X_n$ denote the d-dimensional row vectors consisting of elements drawn from the upper triangles of $CG_{1,t}, CG_{2,t}, \ldots, CG_{n,t}$ respectively. Without loss of generality, assume X_i has zero mean. The original data matrix can be written as $X = [X_1, X_2, \ldots, X_n]$, which is of size $d \times n$. After projection (line 6-9 in Algorithm 2), we perform K-Means clustering on the columns of projected data matrix $Z = [Z_1, Z_2, \ldots, Z_n]$. The confidence of Z_i to be suspicious is proportional to its distance to the nearest center. The details of spatial detection is shown in Algorithm 2. Besides PCA, other techniques such as manifold learning and compressive sensing can also be used here.

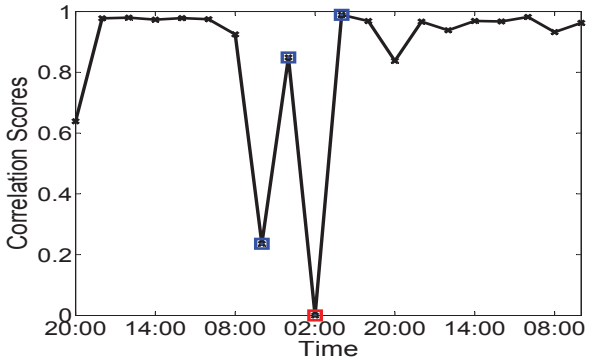
IV. Evaluation

In this section, we provide both case studies and experiments to show the effectiveness of AD.

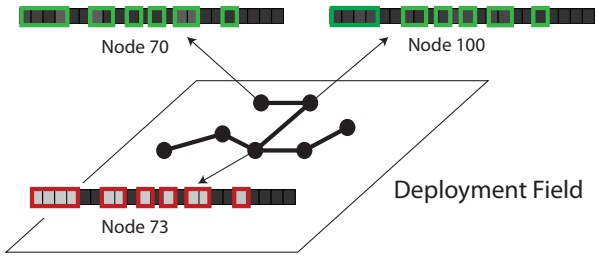
A. Case Studies

We use data collected from 330 nodes in three months from GreenOrbs project. Using AD, we have identified four types of failures.

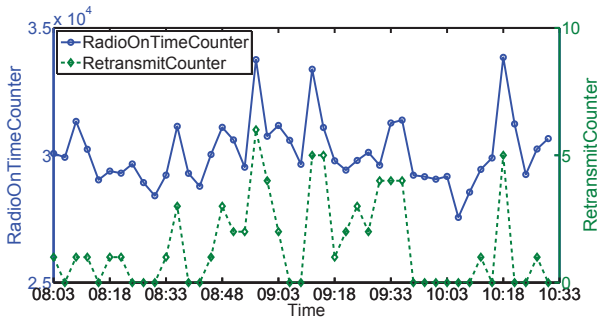
The first type is *ingress drops*, which is detected by correlating ReceiveCounter and TransmitCounter. Since the sampling rate of a sensor is fixed (i.e., three packets every fifteen minutes), the number of packets transmitted by a healthy node should be very close to the number of packets it receives from its children in the routing tree. However, if ingress drops occur, a portion of the incoming packets is dropped.



(a) Correlation scores of TransmitCounter and ReceiveCounter.



(b) Rows of correlation graphs for nodes 70, 73 and 100.



(c) Time series of RadioOnTimeCounter and RetransmitCounter.

Fig. 4. Case studies.

Consequently, a change point in the time series of correlation score between ReceiveCounter and TransmitCounter suggests an ingress drop. As illustrated in Fig. 4a, the correlation score has a significant change point at 2:00. An inspection of the raw data confirms that at that time, this sensor node received packets from its children but did not forward all of them, causing ingress drops. However, if we investigate ReceiveCounter and TransmitCounter separately, this failure cannot be detected since the values of both metrics are within the normal range.

The second type is *routing failures*. A typical symptom is a node changing its parent too frequently. Fig. 4b depicts one row of the correlation graph for three sensor nodes. Each cell in the row stands for the correlation score between ParentChangeCounter and another metric. Note that node 73 has a quite different pattern compared to the others. As an example, consider the second cell in the row, which corresponds to the correlation between ParentChangeCounter and RadioOnCounter. For healthy nodes 70 and 100, the two metrics are not correlated, as they signify the number of times

a node turns on the radio and the number of times it changes its parent node in the routing tree, respectively. For node 73, the correlation score is so high that almost every time it turns on its radio, a parent change happens. Again, if we merely focus on the values of the two metrics independently, such failures cannot be found.

The third type is *link failures*. The detection of link failures exploits the correlation between RetransmitCounter and other metrics such as RadioOnTimeCounter. Fig. 4c illustrates the time series of these two metrics. Initially, they are weakly correlated, since a node may retransmit some packets when its radio is on, but it does not always happen. Starting from 8:30, the correlation score increases, indicating that a large portion of radio-on time is used to retransmit packets. Thus, the transmission from this node to its parent suffers from poor link quality. On the other hand, it is difficult to simply set a threshold for the ratio of retransmitted packets because it may change depending on the time and the location. For instance, a threshold equal to 0.5 may be acceptable in an indoor environment where links are easily interfered, but too low in an open space where link quality is very good.

The fourth kind refers to *node failures* that may be caused by software or hardware problems. The correlation between TaskPostCounter (i.e., the number of tasks posted in TinyOS) and TaskExecuteCounter (i.e., the number of tasks actually performed) can be employed to uncover software faults. We do not address hardware failures because they can be readily detected through a simple rule: when a node crashes, it disappears from the neighbor tables of nearby sensors.

In summary, among the failures discussed in this subsection, only hardware failures can be detected by some simple rules, while other failures require AD.

B. Effectiveness

Temporal Detection: We depict both packet delivery ratios (PDRs) and temporal detection results in Fig. 5. PDR is defined as the ratio between the number of packets received by the sink and that sent by the source node. For each point in the curve, its x-axis value corresponds to the time, while its y-axis value corresponds to the packet delivery ratio at that time. Each square in the figure stands for the time when a change point is reported. We can observe that nearly every square corresponds to a local minimum or maximum in PDRs. It means that the temporal detection results confirm to the variation of system performance. However, some local minima/maxima are not captured by squares, possibly because they do not necessarily signify failures. For example, if a node is healthy but its parent is faulty, it also causes its PDRs to produce a local minimum. Moreover, temporal detection also captures the time when a node recovers from temporary failures, i.e., the squares that correspond to local maxima in PDRs.

Spatial Detection: To validate the effectiveness of spatial detection, we randomly pick a piece of data and investigate 30 faulty nodes that are identified in this phase. Specifically, 23 of them are flagged because failures take place, while 5 of them are false alarms. The other 2 nodes exhibit a sudden increase in the number of packets received, leading to different

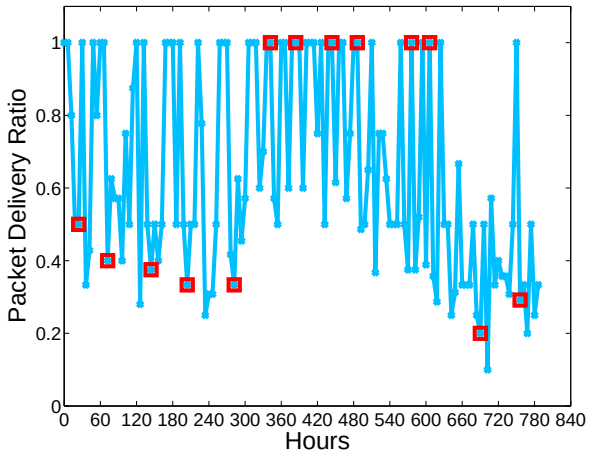


Fig. 5. Packet delivery ratios over time.

correlation scores between PacketReceiveCounter and other metrics. However, after manually inspecting the raw data, we believe that these two nodes are healthy since the number of packets transmitted by them increases accordingly. The involved faults include packet loops, frequent parent changes, link failures, mismatch between radio-on time and packets transmitted/received, ingress failures, and stack overflows. These failures are not independent because several types of failures may occur in the same sensor node simultaneously. Packet loop is the most frequent type of failures.

Overall: We also examine the overall detection ratio and false alarm rate of our approach using a test bed. We vary the network size from 10 nodes to 50 nodes. With different network sizes, we randomly select 10% to 15% of the nodes and inject two kinds of bugs to them respectively.

Fig. 6 plots results of case one in which the software bug causes selected nodes to send each packet multiple times as if they do not receive ACKs from their parent nodes. According to Fig. 6, with small network sizes the detection ratio is almost 100%. As network size grows, the detection ratio tends to decrease and false positive rate increases. The reason is that with small network sizes, only one or two nodes keep sending duplicated packets. Their low volume traffic does not affect other nodes. However, as the number of faulty node increases, the growing number of duplicated packets may lead to congestions in other nodes and raise the false positive rate.

In the second case, we mimic the behavior of ingress drops by reducing the receiving queue length. It can be seen from Fig. 7 that similar trends can also be found as in case one. The reason is that as the number of faulty nodes increases with network size, their child nodes cannot deliver packets. Even if these child nodes switch to other parents, the high volume of data will still cause congestions in the new parent nodes.

V. Related Work

Diagnosis of WSNs has been tackled from various perspectives. The first type aims at software debugging. Clairvoyant [11] is a GDB-like source-level tool that provides a suit of standard debugging commands such as break, set, watch and backtrace. Declarative Tracepoint [12] supports a

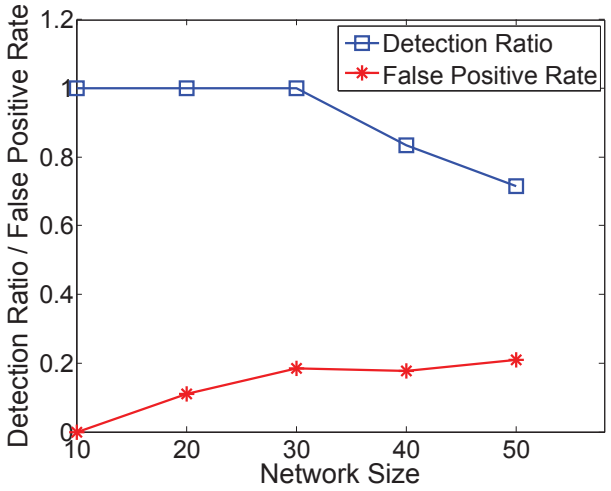


Fig. 6. Detection ratio and false alarm rate of AD when injecting software bugs.

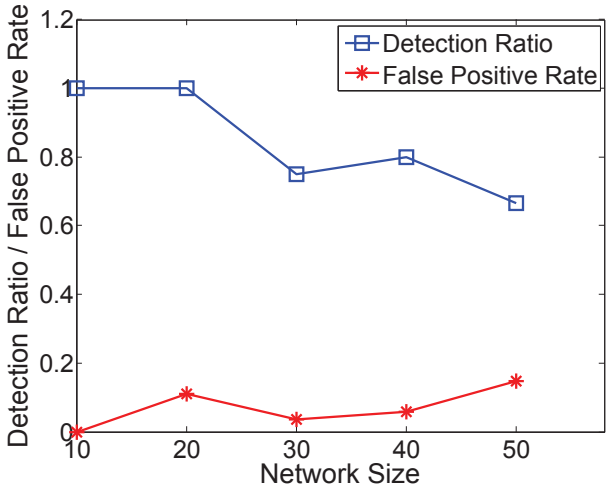


Fig. 7. Detection ratio and false alarm rate of AD when injecting ingress drops.

declarative, SQL-like language, allowing developers to insert a set of action-associated rules to applications at runtime. Our approach differs in that we do not debug the source code. Instead, our goal is to discover the silent failures caused by nodes, ambient environment, and protocols.

The second type of diagnosis is achieved through rule-based mining, or specific inference models. Sympathy [13] periodically gathers information such as nodes' next hops and then builds decision trees to analyze the root causes. DustMiner [14] collects runtime event logs and performs frequent pattern mining to uncover root causes of performance anomalies. PAD [15] leverages a packet marking strategy for constructing and maintaining the inference model. PowerTracing [16] employs a special power meter to identify patterns of power consumption. Unlike these approaches, our work addresses the problem when there is little domain knowledge.

There are also studies closely related to diagnosis [17]–[25]. For example, [18] and [19] focus on data faults. [20] provides an experimental study on link performance. LiveNet [21] uses passive monitoring to reconstruct dynamics of live sensor networks. PD2 [22] pinpoints the root causes of application performance problems. SNMS [23] proposes a sensor network management system.

VI. Conclusion

This paper presents Agnostic Diagnosis, a novel approach that relies on minimum domain knowledge to detect faulty nodes. AD is a sink-based scheme and requires sinks to collect data from all sensors. Consequently, there might exist a delay between fault time and detection time. In the future, we will extend AD such that it can work in a distributed manner. Thus both energy consumption and detection delay can be further reduced. We will also address node mobility [26] in diagnosis.

Acknowledgment

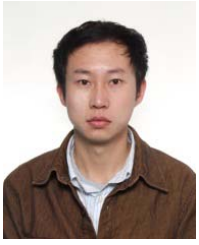
This work is supported in part by the NSFC Distinguished Young Scholars Program under Grant 61125202, National Basic Research Program (973 program) under Grant 2014CB347800, GRF 618011 from Hong Kong RGC, NSFC under Grant 61170213, and NSFC Young Scholar 61103187.

References

[1] M. Li, Y. Liu, J. Wang, and Z. Yang, "Sensor network navigation without locations," in *Proc. IEEE INFOCOM*, 2009.
[2] L. Mo, Y. He, Y. Liu, J. Zhao, S.-j. Tang, X.-Y. Li, and G. Dai, "Canopy closure estimates with greenorbs: Sustainable sensing in the forest," in *Proce. ACM SenSys*, 2009.
[3] G. Werner-Allen, S. Dawson-Haggerty, and M. Welsh, "Lance: Optimizing high-resolution signal collection in wireless sensor networks," in *Proc. ACM SenSys*, 2008.
[4] L. Wang and W. Liu, "Navigability and reachability index for emergency navigation systems using wireless sensor networks," *Tsinghua Science Technol.*, 2011.
[5] M. E. Basseville and I. V. Nikiforov, *Detection of Abrupt Changes: Theory and Application*. Prentice Hall Englewood Cliffs, NJ, 1993.
[6] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proc. fifth Berkeley Symp. Mathematical Statistics Probability*, 1967.
[7] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?" in *Proc. ICDT*, 1999.
[8] I. Jolliffe, *Principal Component Analysis*. Wiley Online Library, 2005.
[9] C. M. Bishop and N. M. Nasrabadi, *Pattern Recognition and Machine Learning*. Springer NY, 2006.
[10] H. Zha, X. He, C. Ding, M. Gu, and H. D. Simon, "Spectral relaxation for k-means clustering," in *Proc. NIPS*, 2001.
[11] J. Yang, M. L. Soffa, L. Selavo, and K. Whitehouse, "Clairvoyant: A comprehensive source-level debugger for wireless sensor networks," in *Proc. ACM SenSys*, 2007.
[12] Q. Cao, T. Abdelzaher, J. Stankovic, K. Whitehouse, and L. Luo, "Declarative tracepoints: A programmable and application independent debugging system for wireless sensor networks," in *Proc. ACM SenSys*, 2008.
[13] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin, "Sympathy for the sensor network debugger," in *Proc. ACM SenSys*, 2005.
[14] M. M. H. Khan, H. K. Le, H. Ahmadi, T. F. Abdelzaher, and J. Han, "Dustminer: Troubleshooting interactive complexity bugs in sensor networks," in *Proc. ACM SenSys*, 2008.
[15] Y. Liu, K. Liu, and M. Li, "Passive diagnosis for wireless sensor networks," *IEEE/ACM Trans. Netw.*, 2010.
[16] M. M. H. Khan, H. K. Le, M. LeMay, P. Moinzadeh, L. Wang, Y. Yang, D. K. Noh, T. Abdelzaher, C. A. Gunter, J. Han *et al.*, "Diagnostic powertracing for sensor node failure analysis," in *Proc. ACM IPSN*, 2010.
[17] M. M. H. Khan, L. Luo, C. Huang, and T. Abdelzaher, "Snts: Sensor network troubleshooting suite," in *Proc. IEEE DCOSS*, 2007.
[18] S. Guo, Z. Zhong, and T. He, "Find: Faulty node detection for wireless sensor networks," in *Proc. ACM SenSys*, 2009.

[19] R. Rajagopal, X. Nguyen, S. C. Ergen, and P. Varaiya, "Distributed online simultaneous fault detection for multiple sensors," in *Proc. ACM IPSN*, 2008.
[20] S. Lin, G. Zhou, K. Whitehouse, Y. Wu, J. Stankovic, and T. He, "Towards stable network performance in wireless sensor networks," in *Proc. IEEE RTSS*, 2009.
[21] B.-R. Chen, G. Peterson, G. Mainland, and M. Welsh, "Livenet: Using passive monitoring to reconstruct sensor network dynamics," in *Proc. IEEE DCOSS*, 2008.
[22] Z. Chen and K. G. Shin, "Post-deployment performance debugging in wireless sensor networks," in *Proc. IEEE RTSS*, 2009.
[23] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler, "Design of a wireless sensor network platform for detecting rare, random, and ephemeral events," in *Proc. ACM IPSN*, 2005.
[24] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in *Proc. ACM SenSys*, 2003.
[25] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin, "Emstar: A software environment for developing and deploying wireless sensor networks," in *Proc. USENIX ATC*, 2004.
[26] X. Wang, X. Lin, Q. Wang, and W. Luan, "Mobility increases the connectivity of wireless networks," *IEEE/ACM Trans. Netw.*, 2013.

Xin Miao received his BS degree in Computer Science and Technology Department from Tsinghua University, China, in 2005, and his Ph.D. degree in Computer Science and Engineering from Hong Kong University of Science and Technology in 2013. He is now a postdoc researcher in the School of Software, Tsinghua University, China. His research interests include sensor networks and RFID.



Kebin Liu received his BS degree in Department of Computer Science from Tongji University, in 2004, and the MS and Ph.D. degrees in Shanghai Jiaotong University, in 2007 and 2010. He is currently an assistant researcher in the School of Software and TNLIST, Tsinghua University. His research interests include sensor networks and distributed systems.

Yuan He received his BE degree in University of Science and Technology of China, his ME degree in Institute of Software, Chinese Academy of Sciences, and his PhD degree in Hong Kong University of Science and Technology. His research interests include sensor networks, peer-to-peer computing, and pervasive computing. He is a member of the IEEE and ACM.

Dimitris Papadias is a Professor of Computer Science and Engineering, Hong Kong University of Science and Technology. Before joining HKUST in 1997, he worked and studied at the German National Research Center for Information Technology (GMD), the National Center for Geographic Information and Analysis (NCGIA, Maine), the University of California at San Diego, the Technical University of Vienna, the National Technical University of Athens, Queen's University (Canada), and University of Patras (Greece). He serves or has served in the editorial boards of the VLDB Journal, IEEE Transactions on Knowledge and Data Engineering, and Information Systems. He was the Program Chair of SIGMOD 2013.

Qiang Ma received his BS degree in the Department of Computer Science and Technology from Tsinghua University, China, in 2009, and his Ph.D. degree in Computer Science and Engineering at Hong Kong University of Science and Technology in 2013. He is now a postdoc researcher in the School of Software, Tsinghua University, China. His research interests include sensor networks, network diagnosis and management and mobile computing.

Yunhao Liu received his BS degree in Automation Department from Tsinghua University, China, in 1995, and an MS and a Ph.D. degree in Computer Science and Engineering at Michigan State University in 2003 and 2004, respectively. He is now Chang Jiang Professor and Dean of School of Software, Tsinghua University, China.