2014 IEEE 34th International Conference on Distributed Computing Systems

Enhancing Visibility of Network Performance in Large-scale Sensor Networks

Xiaoxu Li*, Qiang Ma†, Zhichao Cao†, Kebin Liu†, Yunhao Liu†
*Department of Computer Science and Engineering, The Hong Kong University of Science and Technology
†TNLIST, School of Software, Tsinghua University
Email: lixiaoxu@cse.ust.hk, {maq,caozc,kebin,yunhao}@greenorbs.com

***Abstract*—Being embedded in the physical world, wireless sensor networks (WSNs) present a wide range of failures, due to environment conditions, hardware limitations and software uncertainties, and so on. Once deployed, the interactivity of a WSN greatly decreases, which leads to limited visibility of network performance for managers to investigate sensor behaviors. Existing evidence-based approaches aim to explain particular network symptoms based on expert knowledge and heuristic experiences, which degrade diagnosis accuracy and perform unreliably. These diagnosis models define a limited group of network failures, emphasizing on expert knowledge too much, and thus fail to be adopted to different applications. In this work, we propose VN2, a novel tool to enhance the visibility of network performance. VN2 quantifies a node's state in terms of variation of 43 metrics, and trains a *representative matrix* of network exceptions with Non-negative Matrix Factorization (NMF) model. With this matrix, when a new network state coming up, VN2 automatically attributes abnormal symptoms to one or more root causes. We implement VN2 on testbed and real system traces. Experimental results show that VN2 models network exceptions involving small subsets of root causes, and the interpretation of root causes help us understand network behaviors in details.**

I. INTRODUCTION

WSNs act as increasingly attractive means to bridge the gap between the physical and digital world. Many substantive applications are emerging in recent years, such as health monitoring[8], environmental surveillance[21][23], emergency navigation[20]. The performance of WSN protocols and applications can be efficiently measured in testbed or small-scale networks. Unfortunately, once being deployed in a large-scale exoteric environment, WSNs present a wide range of failures and bugs. Meanwhile, network interactivity and visibility greatly decrease, so it becomes difficult to observe and detect exceptions when they occur.

In the past years, many research studies are presented on diagnosis and measurement for real-world WSN systems. Most existing models, such as Sympathy[16], DustMiner[6], and PAD[11], are inspired from large enterprise diagnosis models (e.g., SNMP[3], Shrink[5]). These methods are evidence-driven, in which people build up inference models for observed symptoms, like decision tree, and Bayesian model, based on their expert knowledge. Two important drawbacks make existing models fall short: 1) These approaches usually assume that an abnormal symptom is caused by one and only one root cause. For instance, Sympathy ranks the possible root causes based on a decision tree. Once a root causes is checked (i.e. predefined threshold is satisfied), the diagnosis process stops. In practice, however, network behaviors are complex and a failure is a combination manifestation of several root causes. 2) Based on expert knowledge and heuristic studies, limited root causes are supposed to impact the whole network. As a result, this pre-knowledge narrows the diagnosis judgment to troubleshoot the real root causes.

Can we measure and model network performance without pre-knowledge or evidences? Agnostic diagnosis[15] provides an inspired idea to this problem. By injecting different types of metrics into a network, the authors propose an outlier detection approach using change point analysis. However, this work targets coarse-granularity diagnosis. It can tell whether a node performs good or not, but fails to provide meaningful explanations to different types of outliers. In response to the limitations mentioned above, we design a measurement and analysis tool, called VN2, which can help understand network behaviors and statuses. In our approach, we pro-actively inject performance correlated metrics $P = \{C_1, C_2, ..., C_M\}$ ($M = 43$) into each node, and these metrics are embedded to continuously monitor a node's state. For example, `NOACK_retransmit_counter` is a time increasing metric, and records the number of retransmit packets due to no successful ACK returned. Two possible root causes explain the metric variation. One possibility is receiver's buffer overflows. The other is that the link quality degrades between sender and receiver. In our testing system, nodes send back three kinds of packets containing the monitoring metrics periodically. For a certain node v, let $P_i^v = \{C_{i,1}^v, C_{i,2}^v, ..., C_{i,M}^v\}$ denote a successive packet. The network state vector of v is the variation of vector $S_v = (s_{i,1}^v, s_{i,2}^v, ..., s_{i,M}^v) = P_i^v - P_{i-1}^v$. Given a node's network state, our goal is to identify the exact root cause (or root causes) occurring at that moment, and quantize the influence of each one.

We introduce Non-negative Matrix Factorization (NMF)[17] model to extract a *representative matrix* from system history logs. After taking a large amount of network states as input, the model outputs a compressed exceptions matrix Ψ, of which each row vector is a potential root cause of a given network exception. When a new network state comes up, we can

1063-6927/14 $31.00 © 2014 IEEE
DOI 10.1109/ICDCS.2014.49

TABLE I
A SAMPLING OF SYSTEM-LEVEL METRICS THAT CORRELATED HAZARD EVENTS IN OUR SYSTEM.

Metric	Potential hazard events	Related network performance
Temperature	Hardware clocks are unstable, due to temperature variation.	Sending packet ratio is controlled by a node's hardware clock. An unstable clock can cause a node sending packets too fast or too slow, which potentially leads to network contention.
Voltage	A node stops working, if its voltage is below 2.8V.	A node cannot send or forward packets. if the node is a key node in a network, the event leads to a breakdown of some subnetworks.
NeighborNum	A node has large subtrees, which means plenty of nodes make the same node as their parents.	The node with large subtrees becomes a key node in a network, and a key node breakdown can cause a great packet loss.
NeighborRssi	A node detect that its neighbors' noises are increasing.	Noises can degrade the packet receive ratio, and also indicates bad link quality.
Overflow_drop_counter	A node's receiving queue overflows.	A queue overflow leads to packet loss, both including incoming and self transmit packets.
NOACK_retransmit_counter	Retransmit a packet because no successful ACK is received.	The link quality between sender and receiver is poor. Or the receiver is not able to deal with the incoming packets.
Parent_change_counter	A node changes its parent frequently.	A frequent parent change indicates a great link dynamics. Sometimes it is quite correlated to environmental conditions.
Loop_counter	A loop appears in a network.	A loop can cause a great packet loss, and energy consumption in a certain area.
Drop_packet_counter	Drop a packet, when it has been retransmitted for 30 times.	The link quality between the sender and receiver can be very poor, or event the sender and receiver are disconnected.
Duplicate_counter	Too many duplicate packets in a network.	Duplicate packets can waste a node energy and storage space, and it also indicates poor link quality.

identify the correlated root causes in Ψ, and quantize each of them. To verify our approach, we implement our approach in both testbed experiments and field studies of a large-scale sensor network system, CitySee. The main contributions are summed up as follows:

- A novel tool, called VN2, is proposed to enhance the visibility of network performance. With VN2, complex network statuses can be disassembled into a compressed and representative matrix Ψ.
- Each row of *representative matrix* Ψ represents a potential root cause for a given network exception. By analyzing the reassembled root causes in Ψ, we can grub detailed understanding of an interested network state.
- Extensive testbed experiments and trace-driven studies are conducted to verify our approach. The results show that our approach is feasible and accurate to measure the network performance of large-scale sensor networks.

The rest of the paper is organized as follows: diagnosis and measurement related work is illustrated in section 2. We present the framework and design details of model in section 3 and 4. We carry out testbed experiments and field studies of CitySee system and illustrate the main results in section 5. In section 6 concludes our work.

II. RELATED WORK

A wide range of applications emerge in WSNs, and most of them inherently rely on persistent and instantaneous sensing data. Hence, it proves necessary to associate sensing work and network management, making the network system more reliable and sustainable. In general, two types of faults would lead to network performance degradation. One is called *function fault*[7][18], including network partition, routing failure, node contention and so on. As we all known, many enterprise-based diagnosis models(e.g. SNMP[3], Shrink[5]) are well-developed for managing devices on IP networks. Similarly, in WSNs, some approaches[4][11][13][19] debug the whole network by collecting information from the distributed nodes. Sympathy[16] is such a prototype tool for detecting and debugging failures in sensor networks. Sympathy has selected metrics that enable efficient failure detection, and triggers failure detection every time it receives a packet from a node, and once a metric period at the very minimum. Failures are detected using flow metrics. Specifically, Sympathy determines whether the sink has received sufficient data from every component on every node over the past epoch. Insufficient data indicates a failure.

Data itself has some time-space correlation. Some approaches[1][22] detect the faults by data mining, e.g., detect some events by localizing the change points. DustMiner[6] looks for sequences of events that may be responsible for faulty behavior, as opposed to localized bugs, such as a bad pointer in a module. Its framework is developed where a front-end collects runtime data logs of the system being debugged and an off-line back-end uses frequent discriminative pattern mining to uncover likely causes of failure. Specifically, DustMiner separates the collected sequence of events into two piles: a "good" pile, which contains the parts of the log when the system performs as expected, and a "bad" pile, which contains the parts of the log when the system fails or exhibits poor performance. A discriminative frequent pattern mining algorithm then looks for patterns that exist with very different frequencies in the two piles. Later, such patterns are analyzed for correlations with preceding events in the logs.

In contrast with those sink-based approaches, there are

some methods[10][12][14] focus on local diagnosis, In [9], the method encourages the sensor nodes to join the fault diagnosis process, as the sensor nodes have the first-hand evidences. In order to avoid to generating additional computation and storage cost to sensor nodes. The authors leverage the fault decision models based on the finite state machine (FSM). A FSM model consist of a finite number of states and transitions between these states. A transition in FSM means a state change when specified condition is fulfilled. After the system is deployed, each sensor continuously monitors its local state information and neighbors. If some exceptions occur, the sensor creates a new FSM model, and broadcast it in the air. Then every neighbor sensor should "push" the state according to its local evidence. When an accepted state is reached, i.e., nodes in a local area all agree with a diagnosis report, the FSM stops and generate a diagnosis report.

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. Also, the interactions within the WSNs and the causal dependencies between root causes and symptoms are usually unknown. As a result, silent failures remain undetected. Agnostic Diagnosis[15] exploits the correlations among metrics of each sensor using *correlation graph* that describes the latent status of the node. Such a correlation graph is updated periodically using the node's metrics. By mining through the correlation graphs, it helps identify the underlying rules of a normally running system, and detect abnormal correlations.

III. Framework

In a real application of WSN, every node executes a serial of commands on the basis of diverse protocols and schemes at different layers. In practice, however, due to many unpredictable reasons such as environmental factors, wrong network configurations, not all the network functions perform well thus abnormal behaviors lead to the breakdown of a node even a network as a whole. To understand what plagues applications in sensor networks, we pro-actively inject a number of metrics into sensor nodes, and the variations of these metrics can be seen as network statuses. We can study and understand network behaviors by analyzing the statuses. Table I shows some hazard events correlated with these metrics that we find interesting in our system logs.

A. System Overview

We first present some basic conceptions as follows:

- **Event**, defined as an exception condition occurring in the operation of hardware or software of a wireless sensor network.
- **Root cause**, comprises a class of network events that can cause other events but are not themselves caused by other events.

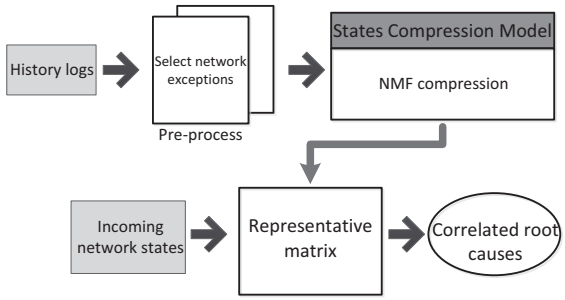


Fig. 1. This study explores the framework of our system: The history logs are used to abstract network exceptions. When network exceptions are sent to states abstraction model, it outputs compression states, stored in a representative matrix, which can be used to reassemble the original network states. For a new incoming network state, the matrix is used to quantize the influence of each correlated root cause.

- **Error**, defined as a discrepancy between an observed, or measured value and a true, specified, or theoretically correct value. Error is a consequence of faults.
- **Symptom**, external manifestation of errors. They are observed as alarms — notifications of potential errors.

When abnormal symptoms are observed, there must be some errors accompanied. Coarse-grain diagnosis can be completed with symptom observations, while fine-grained diagnosis needs to analyze detailed in-network information. For instance, one of the most critical elements to judge whether a network performs well or not is Packet Receive Ratio(PRR) at the sink. It proves trivial for system managers to monitor a node's PRR. Nevertheless, once PRR obtained is distinguished from our expectation, we lack of enough and critical evidences to diagnose the network.

Different from DustMiner[6] which logs network events, our tool embeds system metrics in the protocols running on the sensor nodes. These metrics are triggered to present different variations when network failures happen. It is noted that, one kind of failure can be caused by many possible root causes. For instance, `NOACK_retransmit_counter` is used to count the events when the sender retransmits due to no ACK packet received. This failure can be explained with at least three reasons. One is the receiver's receiving buffer overflows thus it drops the incoming packet. Otherwise, either the data packet is lost in the link from sender to receiver, or the ACK packet is lost in the inverse link. To distinguish different cases and provide more detailed evidences for cross-validation, we also design two other metrics `Drop_packet_counter` and `Duplicate_counter`. The former counter is triggered when a packet is dropped due to receiving buffer overflows, while the latter counts the duplicate packets. If the link from sender to receiver is good, `Duplicate_counter` on the receiver must increase because the sender continuously retransmits the packet. Figure 1 shows the framework of VN2. VN2 is consisted of two main modules. One is *representative matrix* Ψ extraction process. In this module, taking history logs

as input, the output is a matrix, a row vector of which represents one potential root cause of a given network exception. The other component is an inference process. When incoming a state vector of a node, we can identify the possible events happen at this moment, and quantize each of the correlated root causes.

B. Problem Statement

As mentioned in many literatures[6][11], a network exception can be triggered by many root causes simultaneously and repeatedly. This phenomenon proves that, it is inaccurate and unreasonable to explain an occurred network exception with one exact root cause in all cases. By collecting anomaly exceptions and learning with our model, we aim to achieve three targets: 1)extract the *representative matrix*, each row vector denotes a root cause; 2)given a state vector of a node, besides judging whether it functions well or not, we also identify the exact root cause (or root causes) occurring at that moment; 3)quantize the influence for each root cause, so we can rank them accordingly, and thus make the recovery work more efficient.

To a certain node v, we denote two successive packets (node states) as $P_i^v = (C_{i,1}^v, C_{i,2}^v, ..., C_{1,M}^v)$ and $P_{i+1}^v = (C_{i+1,1}^v, C_{i+1,2}^v, ..., C_{i+1,M}^v)$, in CitySee $M = 43$. Ideally, if we ensure that one and only one root cause $\mathcal{E}$ happens during this interval, we can say that $\mathcal{E}$ leads to such a variation of vector $F_v = (C_{i+1,1}^v - C_{i,1}^v, C_{i+1,2}^v - C_{i,2}^v, ..., C_{i+1,M}^v - C_{i,M}^v)$. Inversely, to any node, if we observe that its state changes nearly the same as F_v, we can say that $\mathcal{E}$ happens on node v. In our system, we define N root causes $\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_N$ for a sensor node. Specifically, due to network topology properties, nodes can have several normal states. Moreover, we ensure that every exception state will trigger some but not all root causes. Based on 7 days realistic data and heuristic analysis, we expect to obtain a *representative matrix* between network states and metrics, which is represented by Ψ:

$$\mathbf{\Psi} = \begin{pmatrix} \psi_{1,1} & \psi_{1,2} & \cdots & \psi_{1,N} \\ \psi_{2,1} & \psi_{2,2} & \cdots & \psi_{2,N} \\ \vdots & \vdots & \vdots & \vdots \\ \psi_{2,1} & \psi_{2,2} & \cdots & \psi_{M,N} \end{pmatrix}$$

where $\psi_{i,j}$ is the variance of system metric i when only root cause j happens, for $i = 1, 2, ..., M$, and $j = 1, 2, ..., N$. The row of Ψ^T, a root cause, can be referred to as $\psi_j = [\psi_{1,j}, \psi_{2,j}, ..., \psi_{M,j}]^T$, $j = 1, 2, ..., N$, which represents the node state variance to root cause j, where the superscript T denotes transposition.

Our tool is based on a front-end that reports network metrics and a back-end that analyzes it. To diagnose a node v, we denote v's state S_i^v by subtracting two successive packets P_{i-1}^v and P_i^v: $S_i^v = (s_{i,1}^v, s_{i,2}^v, ..., s_{i,M}^v) = (C_{i,1}^v - C_{i-1,1}^v, C_{i,2}^v - C_{i-1,2}^v, ..., C_{i,M}^v - C_{i-1,M}^v)$, $i > 0$. Given the above *representative matrix*, we have:

TABLE II
NOTIONS IN OUR APPROACH.

Notions	Descriptions
P	Set of metrics injected in CitySee system.
S_v	A node's network state, containing various network metrics.
E	Network exceptions.
Ψ	Representative matrix. Each row vector is regarded as a root cause.
W	The correlation strength between exceptions and root causes.
r	Compression factor. Also, its the number of possible root causes.

$$S_i^v = \Psi X = \begin{pmatrix} \psi_{1,1} & \psi_{1,2} & \cdots & \psi_{1,N} \\ \psi_{2,1} & \psi_{2,2} & \cdots & \psi_{2,N} \\ \vdots & \vdots & \vdots & \vdots \\ \psi_{2,1} & \psi_{2,2} & \cdots & \psi_{M,N} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}$$

where $x_j \geq 0$, $j = 1, 2, ..., N$. $x_j \neq 0$ if and only if the root cause ψ_j does happen on node v at this moment. In most cases, the node performs well, such that the variation $x_j \approx 0$ for $i = 2, 3, ..., N$. Otherwise, x_j can be explained as the times ψ_j impacts v. In the next section, we give detailed process about how to build matrix Ψ. The notions used in the paper is in Table II.

C. Data Sources

To set context of our analysis, we briefly introduce our system and detailed data sources.

CitySee Network. An ongoing urban carbon dioxide sensing project, and mainly aims to monitor concentration of carbon dioxide. It employs the TelosB mote with msp430f1611 processor and CC2420 radio. A single TelosB sink node is used for collecting data. In this paper, we collected 7 days data from Aug. 1 2011, including 286 nodes and 145,571 packets in total.

System Metrics. The sink collects three types of packets with CTP collection protocol, denoted as C1, C2, and C3. We introduce the metrics used in our system. The C1 packet contains two kinds of information: (1) sensor data, including temperature, humidity, light, and voltage; (2) routing information, including path-ETX from the source node to the sink node, and node IDs along the path.

The C2 packet contains the routing table with a maximum neighbor number of 10. Each routing table entry contains (1) the neighbor node ID; (2) the RSSI value of each neighbor; (3) the link-ETX estimation to the neighbor (4) the path-ETX estimation to the link.

The C3 packet contains various counters: (1) the `Parent_change_counter` counter to record a node's parent change event. (2) the `Transmit_counter` records the accumulated number of transmitted packets (3) the `Receive_counter` counter records the accumulated number of received packets (4) `Overflow_drop_counter`

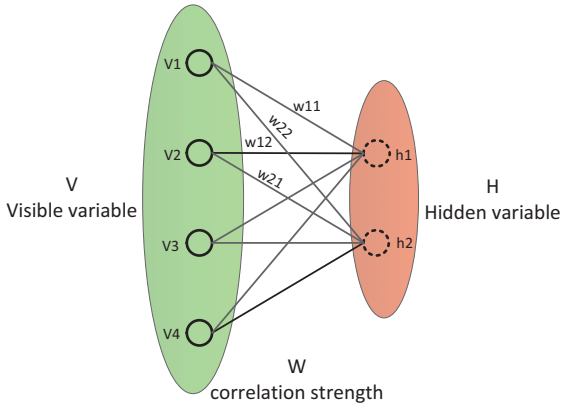


Fig. 2. Probabilistic hidden variable underlying NMF. The model is diagrammed as a network. According to the model, the visible variable v_i is generated from a probability distribution with mean $\Sigma_a W_{ia} h_a$. In the network diagram, the influence of h_a on v_i is represented by a connection with strength W_{ia}.

records the accumulated number of packets drops due to queue overflow (5) the `Loop_counter` counter records the accumulated number of detected loops.

IV. Trace Study and Analysis

This section illustrates several design details of VN2 system. Giving the system logs and data, we first identify all potential circumstances of network exceptions, which are usually hidden from abundant normal data. Then we extract a *representative matrix* Ψ of these network exceptions by Non-negative Matrix Factorization (NMF) model. Each row of *representative matrix* is treated as a root cause vector. Finally we give comprehensive explanation of these root cause vectors by expert knowledge.

A. Non-negative Matrix Factorization

Non-negative Matrix Factorization (NMF) model has been used for static data analysis and pattern recognition. It contains a group of algorithms to factorize matrix V into two matrices W and Ψ, in which all elements are non-negative. Given a set of multivariate n-dimensional data vectors, the vectors are placed in the columns of $n \times m$ matrix V. This matrix is then approximately factorized into an $n \times r$ matrix W and an $r \times m$ matrix Ψ. Matrix Ψ is called *representative matrix*, which is regarded as the probabilistic hidden characteristics of the original data in matrix V. W is the corresponding correlated strength.

Figure 2 illustrates the dependency between representative matrix Ψ and visible matrix V. The factorizations is in the form of $V \approx WH$ or

$$V_{nm} \approx (W\Psi)_{nm} = \sum_{i=1}^{r} W_{ni}\Psi_{im}$$

The dimensions of the matrix factors W and Ψ are $n \times r$ and $r \times m$, respectively. Factor r is usually smaller than n or m and fulfills the constraint $(n + m)r < nm$. In our system, all metrics are positively grown over time. Thus it inspires us to extract the hidden root causes from abundant network data and logs by NMF.

B. Using History Data to Compress Network States

In our approach, we inject 43 metrics into our system, and these metrics are used to monitor node behaviors at physical layer, link layer, network layer and application layer, and all these metrics are non-negative. Each node sends back packets containing these metrics to sink every 10 minutes. For our system trace, we collect 145,571 packets in total during observation from Aug. 1st to Aug. 7th, 2011.

Using NMF model, we can compress all network statuses directly and extract the representative matrix. However, in the most of time, the network works well. The logs of normal network status take the majority part of all logs. Usually, we care more about abnormal network behaviors, which can degrade network performance. The straightforward factorization makes normal statuses conceal representability of network exceptions.

To avoid such a situation, given the history logs, we try to identify all the potential exceptions happened in the logs. We have observed that the variance of all metrics is relatively stable when the system works well. Thus we treat the records with large variance of certain metric as potential exceptions. We calculate the mean value of each injected metric. For all metrics of a certain state u, the variance between u and the mean value are calculated, denoted as ϵ_u. If $\frac{\epsilon_u}{\max(\epsilon_u)} \geq 0.01$, the state u is identified as an exception.

Figure 3(a) shows an exception detection example. `Voltage`, `Neighbor_rssi`, `Radio_on_time`, and `Receive_counter` are four injected metrics. As the figure illustrates, the variation of these metrics changes with time. The most of the variations are around zero, indicating normal statuses. The discrete points are kinds of exceptions correlated with different metrics. After finding all the potential exceptions, we use them as the input of NMF model. Hopefully, the output *representative matrix* can be used to reassembled the characteristics of original network exceptions.

Problem 1 *Let $E_{n\times m}$ denote network exceptions, and n is the number of abnormal statuses, while m denotes the number of metrics. We want to find a factorization, such that*

$$(E_{n\times m}) = W_{n\times r}\Psi_{r\times m} \quad (1)$$

Ψ is called representative matrix*, each row vector of which is a root cause vector. r is the compression factor, and W is the correlation strength, which quantizes the amount of influence of each root cause vector of matrix Ψ.*

Our problem is formalized in Problem 1. Algorithm 1 illustrates the specific solving process. The algorithm takes network exception states E_{nm} and factor r as input, deduces

Algorithm 1 Network States Compression

Input: The long-term history network states $E_{n\times m}$, and the compression factor r;
Output: States factorization $E_{n\times m} \approx W_{n\times r}\Psi_{r\times m}$;

1: Generate random matrixes $W_{n\times r}$ and $\Psi_{r\times m}$;
2: **for** $i = 1$ to r **do**
3: **for** $j = 1$ to m **do**
4: $\Psi_{ij} \leftarrow \Psi_{ij}\frac{(W^TV)_{ij}}{(W^TW\Psi)_{ij}}$;
5: **end for**
6: **end for**
7: **for** $i = 1$ to n **do**
8: **for** $j = 1$ to r **do**
9: $W_{ij} \leftarrow W_{ij}\frac{(V\Psi^T)_{ij}}{(W\Psi\Psi^T)_{ij}}$;
10: **end for**
11: **end for**

Algorithm 2 Basis Matrix Sparse Process

Input: Weight matrix W;
Output: Sparse matrix $\bar{W}$;

1: normalization W
2: Sort all the elements in W in descending order and store in W';
3: Generate a temporal matrix $\bar{W}$;
4: **while** $\|\bar{W}\| < 0.9\|W\|$ **do**
5: Move the maximum element in W'into $\bar{W}$;
6: **end while**

W and Ψ in a iterate process, and outputs Ψ as the *representative matrix*. Using equation of step 4 and 9, theorem 1 ensures the convergence of iteration process. Detailed proof is shown in [17].

Theorem 1 *The Euclidean distance $\|E - W\Psi\|$ is non-increasing under the update rules*

$$\Psi_{ij} \leftarrow \Psi_{ij}\frac{(W^T)_{ij}}{(W^TW\Psi)_{ij}}, \quad W_{ij} \leftarrow W_{ij}\frac{(E\Psi^T)_{ij}}{(W\Psi\Psi^T)_{ij}} \qquad (2)$$

The Euclidean distance is invariant under these updates if and only if W and Ψ are at a stationary point of the distance.

Another problem is how to determine factor r? Two important observations are mainly considered. One observation is that according to Occam's razor principle [2], we want to explain exception state with root cause vectors as few as possible. Thus we want to keep r as small as possible. Meanwhile, W is better to be sparse so that each exception of E is just explained by a few root cause vectors in Ψ. We remove the less influenced correlated strength to get matrix W sparse. The process is shown in algorithm 2. According to step 4, we ensures that the sparse matrix $\bar{W}$ retains 90% information that W holds.

We define the approximation accuracy as following.

Definition 1 *In NMF model, for an original matrix E, it can be factorized into two matrix W and Ψ. The approximation accuracy α is defined as $\alpha = \|E - W\Psi\|$.*

The other observation is that because the NMF factorization is approximation process, the compression accuracy will be increased with r is getting small. Moreover, if we replace W with sparser matrix $\bar{W}$, the compression difference will be increased when r is large. In Figure 3(b), the approximation accuracy is calculated using original W and sparse $\bar{W}$. As the result shows, when r is larger than 30, the sparse matrix $\bar{W}$ holds more difference than original W holds. Meanwhile, when r is smaller than 15, the compression difference increases quickly. Thus we choose r as 25 for our system data. Given history network logs, we want to ensure the original exceptions can be compressed as accuracy as possible. Meanwhile, to give detailed analysis on a network state, we hope to keep as much as possible information in matrix Ψ. Using our system logs, we choose r as 25.

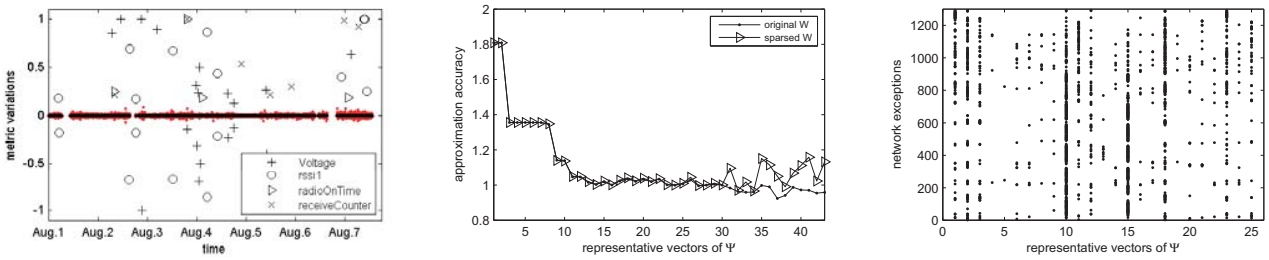
C. Representative Matrix Ψ and Root Cause Vectors Interpretation

After NMF process, the model outputs the *representative matrix* Ψ, which can be used to reassemble original exceptions happened in the network. Since Ψ is a compressed version of original statuses, we are interested in the meanings of each representative vector, namely root cause vector. Here we formalize problem 2.

Problem 2 *Given the representative matrix Ψ, each row vector of which can be regarded as a root cause vector, or a feature of a kind of network exception. We need to label these root causes with comprehensive network interpretation.*

We have 25 root cause vectors in total, and each of these vectors can be explained by comprehensive network behaviors. In Figure 4, we show six root cause vectors of Ψ, and these features can be categorized into three types. Ψ_1 and Ψ_2 are the features correlated with metrics in C1 packet, and these metrics (e.g. humility, light) are mostly effected by surrounding environment. Ψ_3 and Ψ_4 are the vectors recording the RSSI information of a node's neighbors, which influences link quality and dynamics, and Ψ_5 and Ψ_6 are correlated with various protocol counters (e.g. Loop_counter, Parent_change_counter), respectively. We give detailed explanations on these vectors, taking Ψ_2, Ψ_3, Ψ_5 and Ψ_6 for example.

Ψ_2 is a node's voltage. There are two possible explanations for this great variation of this metric. One is that a node reboot happens, and the other is that a node consumes too much energy during the time interval. Ψ_3 is a feature correlated with a node's neighbors' RSSI, and these metrics variations mean a node experiences a dynamic link quality. This phenomena might be caused by environment change, mobile obstacles or co-existing signals.

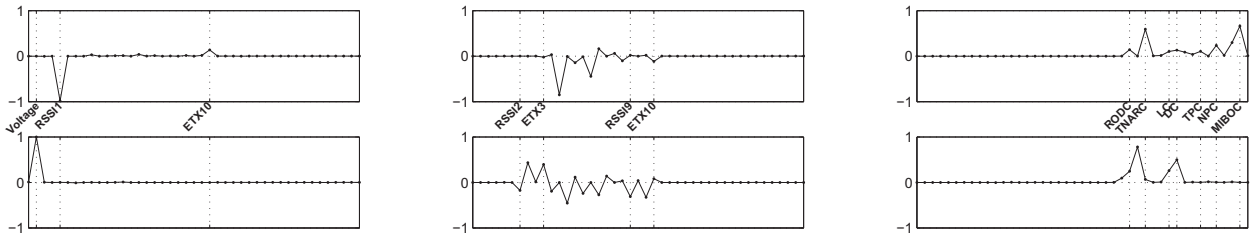


(a) Metric variations over time.

(b) Compression accuracy.

(c) Correlation between each exception and root cause vectors in Ψ.

Fig. 3. *representative matrix* Ψ extraction model and parameter configuration:(a) four types of exceptions, correlated with different metrics (e.g. voltage, Receive_counter etc.). (b) choose factor r using original correlated strength W and sparse $\bar{W}$. (c) given an exception, when there is a point appears in a certain row vector, it means that the exception is correlated with the root cause vector.



(a) Ψ_1 and Ψ_2 correlated with physical factors.

(b) Ψ_3 and Ψ_4 correlated with link quality.

(c) Ψ_5 and Ψ_6 correlated with protocol parameters.

Fig. 4. System representative matrix $\Psi_{25\times 40}$ features example: the x-axis denotes the injected metrics. (a) metrics in C1 packet (e.g. voltage). (b) RSSI and ETX(expected transmission count) information of a node's neighbors. (c) protocol counters (e.g. LC(Loop_counter), DC(Duplicate_counter)).

Ψ_5 and Ψ_6 are both the features correlated with counters in C3 packet. Because different combinations of counters give different root causes inference, we give explanations to Ψ_5 and Ψ_6 as examples. Ψ_5 can be explained as follows: the two counters with great variations are `NOACK_retransmit_counter` and `MacI_backoff_counter`. The later indicates a severe contentions happened in the network, and a node can not send or receive a packet successfully. The most proper explanation is that link quality degradation, and such degradation may causes by environment factors.

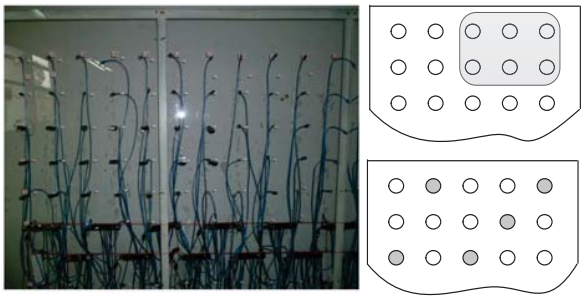
Ψ_6 can be explained as follows: to a node v, when routing loop occurs, for both forwarded and self-generated packets, they are repeatedly sent out and received by node v, until the loop disappears. That is why `Transmit_counter` and `Self_transmit_counter` increase simultaneously. The metric duplicate packets, which are mainly caused in two ways. On the one hand, all nodes in this loop keep receiving a packet for many times, until the source node pro-actively drops it for some other reasons (e.g., any packet is tried to sent out for 30 times at most). On the other hand, the loop imposes restrictions on forwarding the packets in the receiving queue for node v. Intuitively, once the receiving queue overflows, v fails to receive any packet from its child nodes. As a result, without ACK replies, its child nodes keep transmitting the same packets to v. That is also why the metric `Overflow_drop_counter` increase, but not higher than `Duplicate_counter`.

After giving detailed explanations on each root cause vector in the representative matrix Ψ, we can use Ψ for further network state analysis.

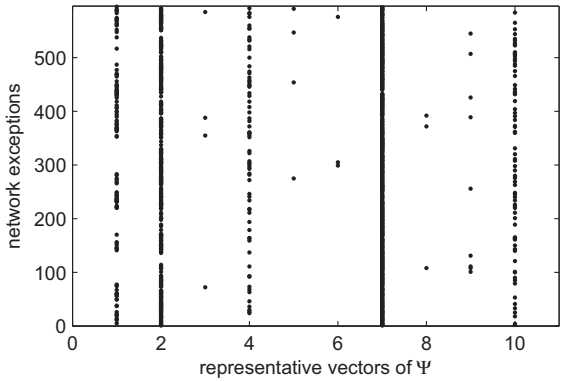
Problem 3 *To a node v, at a certain moment, its status can be denoted as $S_v = (s^v_{i,1}, s^v_{i,2}, ..., s^v_{i,M}) = P^v_i - P^v_{i-1}$ (P_i is a successful packet of node v). Given the* representative matrix Ψ*, we want to calculate its correlation strength w_v to Ψ. The problem $s = w_v\Psi$ can be converted to an optimization problem:*

$$\begin{aligned} &argmin_w (e_v - w_v\Psi)^T (e_v - w_v\Psi) \\ &s.t. \qquad w_i \geq 0 \end{aligned} \tag{3}$$

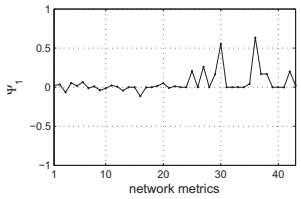
To understand a state vector of a node at a certain moment, as formed in Problem 3, we can identify the exact exceptions and infer the possible root causes that trigger the exceptions, using matrix Ψ. In Problem 3, the optimization problem is a *convex programming* problem. In Figure 3(c), we calculate the correlation between each detected exception and row vectors of Ψ.



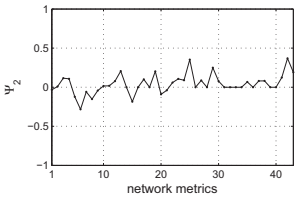
(a) Testbed and two experiments scenarios.



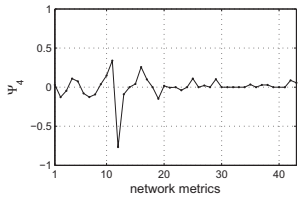
(b) Correlation with row vectors in Ψ



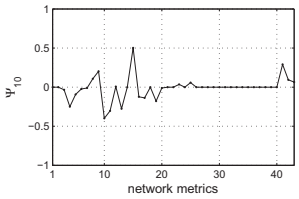
(c) Ψ_1 related metrics variations.



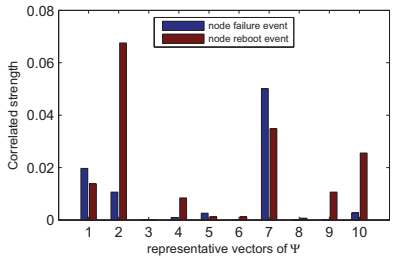
(d) Ψ_2 related metrics variations.



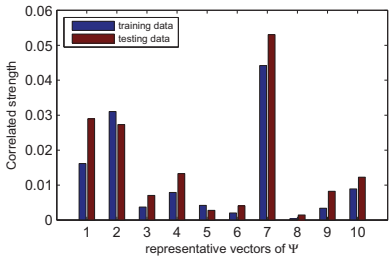
(e) Ψ_4 related metrics variations.



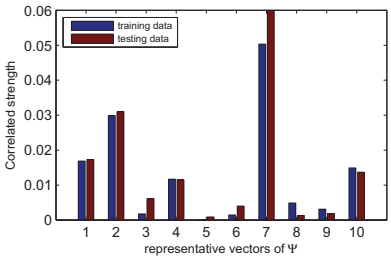
(f) Ψ_{10} related metrics variations.



(g) Root causes distribution of exceptions.



(h) Scenario 1: root causes distribution.



(i) Scenario 2: root causes distribution.

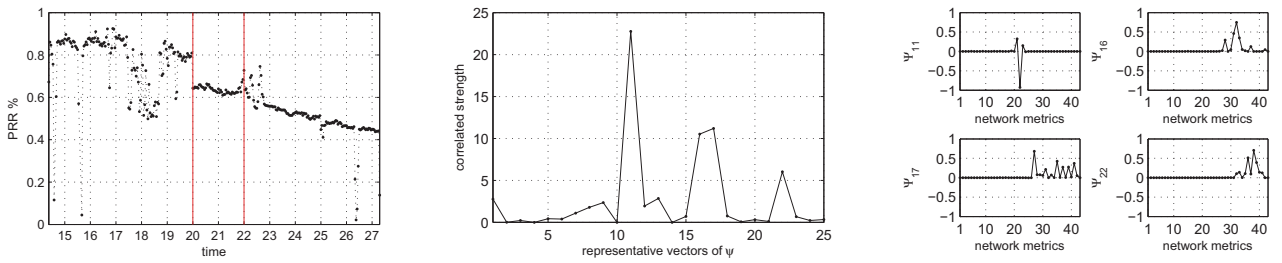
Fig. 5. Testbed experiments setup and results. 45 telosb nodes are used in the experiments. By manually introducing two types of network events, node failure and node reboot, we implement our approach to study and analyze the experiment trace: (a) testbed implementation. (b) extract representative matrix Ψ using one hour testbed trace. (c) to (f) main correlated root causes analysis. (g) correlated strength on two types of exceptions. (h) in scenario 1: when nodes are removed locally, the detected features variation of test data compare with that of training data. (i) in scenario 2: when nodes are removed expansively, the detected features variation of test data compare with that of training data.

V. Evaluation

We implement our model on testbed and real system traces. For the testbed experiments, we manually introduce two types of network exceptions: node failure and node reboot. Compared with the ground truth, we verify the feasibility and accuracy of our model. Furthermore, using the *representative matrix* Ψ extracted in section 4, we analyze a real system trace of CitySee to study and explain the network performance.

A. Testbed experiments

We implement testbed experiments to verify the accurcy of our model. In the experiments, 45 telosb nodes are used with communication power 2 of CC2420 in TinyOS. Each node sends three types of packets C1, C2 and C3 every three minutes. Two types of network events are manually introduced: 1) node failure. It can be set by periodically removing some nodes from testbed. 2) the other is node reboot. This type of event can be set by putting back the removed nodes. In the experiments, we remove 5-7 nodes and put back some of these nodes every 10 minutes. Figure 5(a) shows the testbed, and nodes distributes in a 9×5 matrix area. Two experiment scenarios are designed. One is to remove nodes expansively at a time, and in the other, nodes are removed in local area.



(a) System PRR distribution.

(b) Correlation strength of representative vectors of Ψ of system trace from Sep. 20 to 22.

(c) Detailed information about main features.

Fig. 6. CitySee system trace experiments: (a) A real system trace: PRR distribution from Sep. 14 to 27. (b) using the representative matrix $\Psi_{25\times43}$ we extract in Section IV, the correlations strength between trace data and representative vectors of Ψ are calculated. (c) Detailed features information.

The experiments start from 13:40pm, and last for about two hours, and we get 1,639 packets in total. We use first hour data for training, and the other hour data for testing. Using the training data, we extract the *representative matrix* Ψ. Because the training set is small, and the normal statuses are not large enough to conceal the representation of exceptions, we skip the exception extraction process, and compress all the statuses together. According to the principle of choosing factor r, and for the testbed trace, r is chosen as 10. In Figure 5(b), it shows correlations with row vectors of Ψ of all the training data. As the figure shows, most of the exceptions are correlated with root cause vectors Ψ_1, Ψ_2, Ψ_4, Ψ_7, and Ψ_{10}. Detailed information about these vectors are shown in Figure 5(c) to 5(f).

We give brief explanations to these features. As we can see, vector Ψ_7 is used much more times than any other features. Because the *representative matrix* is compressed using all trace data, feature Ψ_7 must be a representation of normal network states. Also, the inference can be verified by examining metric variations in the feature, and there is no harm variation detected. For vectors Ψ_2 and Ψ_{10}, variations mainly happen on two kinds of metrics, `NeighborRssi` and `NeighborEtx`, which indicate link dynamics around a certain node. To feature Ψ_1, there are two greatly changed metrics `NOACK_retransmit_counter` and `Parent_change_counter`. To a node v, the former metric indicates that the node cannot successfully sending packets to some of its neighbors. The reasons could be either link degradation, or receiver failures. By taking the other metric into account, the node has changed its parent. Based on these observations, the most likely reason is that the parent of the node is not reachable currently. To feature Ψ_4, a peak happens when a node detects a new neighbor joins. The phenomena can be explained as a node reboot event. Our explanations can be verified according to results in Figure 5(g). As the figure shown, the ground truth is that: 1) On the one hand, when features Ψ_1 and Ψ_2 are the main exceptions happened at that moment, the most possible reason is that nodes failures occur in the network. We can combine the information of PRR(Packet Reception Ratio) at that moment to further locate which nodes have failed; 2) On the other hand, besides the variation of features Ψ_1 and Ψ_2, if features Ψ_4 and Ψ_{10} are identified to show variations at the same time, the most likely reason is that some node have rebooted in the network.

For the testing data, we calculate the *correlation strength* for both training and testing data, and results are shown in Figure 5(h) and Figure 5(i) for two scenarios, nodes removed in group and local areas, respectively. The results illustrate that 1) both the training and testing traces mainly use features Ψ_1, Ψ_2, Ψ_4, Ψ_7, and Ψ_{10}, and *correlation strength* values for each features between these two data sets are positively related. Because the two traces have similar network statuses, the representations are also quite alike. This result verifies that our approach is feasible and accurate. 2) we have an interesting observation that result in Figure 5(i) has more accurate representation than results in Figure 5(h). Because such exceptions are easier to be detected, when we remove or put back nodes expansively. In other words, when distinct and large-scaled exceptions happen in the network, our model is able to detect such exceptions accurately.

We also implement VN2 to detect and analysis exceptions on a real system trace from September 15 to 27 of CitySee. The *representative matrix* $\Psi_{25\times43}$ is extracted in previous section. We gather statistics of system PRR, and observe that there is an obvious PRR degradation from Sep. 20 to 22, shown in Figure 6(a). We are interested in what happened in the network during that period, and try to find the possible root causes to explain the observation.

The correlations strength between the representative vectors of Ψ and, all state vectors collected from the trace, are calculated and illustrated in Figure 6(b). As the figure shows, most of the exceptions are correlated with representative vectors Ψ_{11}, Ψ_{16}, Ψ_{17}, and Ψ_{22}. Detailed information about these vectors are shown in Figure 6(c). We give brief explanations to these features: 1) As we mentioned in precious section, feature Ψ_{16} represents a network loop event, that means the state vector detects a loop exception occurring at that moment. We can

extract all loop exceptions to further locate the loop in the network. 2) To feature Ψ_{22}, metrics `No_parent_counter` and `Parent_change_counter` variate more than others. It indicates that node failures happen in the network. Meanwhile, feature Ψ_{11} usually correlated with reboot or link degradation. Both types of exceptions are verified in the testbed experiments. 3) Feature Ψ_{17} have two important metrics variations, `NOACK_retransmit_counter` and `MacI_backoff_counter`. When both features happen, the most proper explanation is that link quality degradation, and such degradation may causes by environment factors. To sum up, we can infer that three main network exceptions occurred during that period: network loop, contention, and node failures, that cause the PRR degradation.

VI. Conclusion

In emerging sensor network applications, it is necessary to find out the exact root cause of a failure in a wireless sensor network. Generally, on the one hand, more information collected, more specific the diagnosis for the network. On the other hand, more information collected also means more delay and overhead. Hence, it is essential to develop algorithms for scenarios in which only limited information is allowed, and design robust system tools to better troubleshoot a WSN. In this paper, we propose VN2, a novel tool to enhance the visibility of network performance. By injecting performance related time-varying metrics into each sensor node, we are able to quantify the node's state in terms of the variation of these metrics. VN2 pro-actively collects the state information from the system, and trains a *representative matrix* of network exceptions with Non-negative Matrix Factorization (NMF) model. With this matrix, VN2 automatically attributes abnormal symptoms to one or more root causes. To verify the feasibility of our approach, we provide testbed experiments and trace-driven field studies of a large-scale sensor networks system. By using the representative matrix, when a new network status comes up, we can calculate the correlated root causes in Ψ, and provide a detailed explanation to the status via the physical meanings of chosen vectors. In the future, we want to further develop VN2 so that it can be used for combination diagnosis, protocol performance estimation, and so on.

Acknowledgement

We thank the anonymous reviews for their constructive comments. This work is supported in part by the NSFC under grant 61103187, and the NSFC Distinguished Young Scholars Program under grant 61125202.

References

[1] S. Ahuja, S. Ramasubramanian, and M. Krunz. Srlg failure localization in all-optical networks using monitoring cycles and paths. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 700–708. IEEE, 2008.

[2] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Occam's razor. *Information processing letters*, 24(6):377–380, 1987.

[3] J. Case, M. Fedor, M. Schoffstall, and C. Davin. *A simple network management protocol (SNMP)*. Network Information Center, SRI International, 1989.

[4] S. Guo, Z. Zhong, and T. He. Find: faulty node detection for wireless sensor networks. In *Proceedings of the 7th ACM conference on embedded networked sensor systems*, pages 253–266. ACM, 2009.

[5] S. Kandula, D. Katabi, and J. Vasseur. Shrink: A tool for failure diagnosis in ip networks. In *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, pages 173–178. ACM, 2005.

[6] M. Khan, H. Le, H. Ahmadi, T. Abdelzaher, and J. Han. Dustminer: troubleshooting interactive complexity bugs in sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 99–112. ACM, 2008.

[7] 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 *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 117–128. ACM, 2010.

[8] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, pages 254–263. IEEE, 2007.

[9] C. Liu, X. Yan, L. Fei, J. Han, and S.P. Midkiff. Sober: statistical model-based bug localization. *ACM SIGSOFT Software Engineering Notes*, 30(5):286–295, 2005.

[10] K. Liu, Q. Ma, X. Zhao, and Y. Liu. Self-diagnosis for large scale wireless sensor networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 1539–1547. IEEE, 2011.

[11] Y. Liu, K. Liu, and M. Li. Passive diagnosis for wireless sensor networks. *IEEE/ACM Transactions on Networking (TON)*, 18(4):1132–1144, 2010.

[12] Q. Ma, K. Liu, X. Miao, and Y. Liu. Sherlock is around: Detecting network failures with local evidence fusion. In *INFOCOM, Proceedings IEEE*, pages 792–800. IEEE, 2012.

[13] E. Magistretti, O. Gurewitz, and E. Knightly. Inferring and mitigating a link's hindering transmissions in managed 802.11 wireless networks. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, pages 305–316. ACM, 2010.

[14] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Analyzing the mac-level behavior of wireless networks in the wild. *ACM SIGCOMM Computer Communication Review*, 36(4):75–86, 2006.

[15] X. Miao, K. Liu, Y. He, Y. Liu, and D. Papadias. Agnostic diagnosis: Discovering silent failures in wireless sensor networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 1548–1556. IEEE, 2011.

[16] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. Sympathy for the sensor network debugger. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 255–267. ACM, 2005.

[17] D. Seung and L. Lee. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, 13:556–562, 2001.

[18] D. Son, B. Krishnamachari, and J. Heidemann. Experimental analysis of concurrent packet transmissions in low-power wireless networks. In *Proc. of the SenSys Conf.[1]*, pages 237–250, 2005.

[19] T. Sookoor, T. Hnat, P. Hooimeijer, W. Weimer, and K. Whitehouse. Macrodebugging: Global views of distributed program execution. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 141–154, 2009.

[20] Y. Tseng, M. Pan, and Y. Tsai. Wireless sensor networks for emergency navigation. *Computer*, 39(7):55–62, 2006.

[21] N. Xu, S. Rangwala, K.K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. A wireless sensor network for structural monitoring. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 13–24. ACM, 2004.

[22] J. Yang, M.L. Soffa, L. Selavo, and K. Whitehouse. Clairvoyant: a comprehensive source-level debugger for wireless sensor networks. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 189–203. ACM, 2007.

[23] Q. Zhang, L. Fu, Y. Gu, L. Gu, Q. Cao, J. Chen, and T. He. Collaborative scheduling in dynamic environments using error inference. pages 105–114, 2011.